LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SOLO PROGRAMADORES

NO VI. 2ª ÉPOCA NÚMERO 62 UNA PUBLICACIÓN DE: REVISTAS PROFESIONALES S.L. 975 Ptas. • 1.280 Esc-Cont • 5,86 € (IVA incluido)

API: API DE TELEFONÍA (II)
uestro proyecto
n Visual Basic

ULTIMEDIA (II)
reación de un reproductor
ultimedia en Delphi IV

EPLHI 5
onoce sus principales
ovedades

OMDEX LAS VEGAS'99
nálisis de la principal
eria informática

INUX: DNS A FONDO (III)
ómo implementar un
ervidor DNS

A TECNOLOGÍA ASP (III)
rofundizando en los formularios

AVA: CRIPTOGRAFÍA (III)
os algoritmos de clave privada

B: SUITE DE INTERNET (IV)
ómo crear un chat

TRANSFERENCIA DE ARCHIVOS PUNTO A PUNTO (V)
Programación de nuestro protocolo de enlace

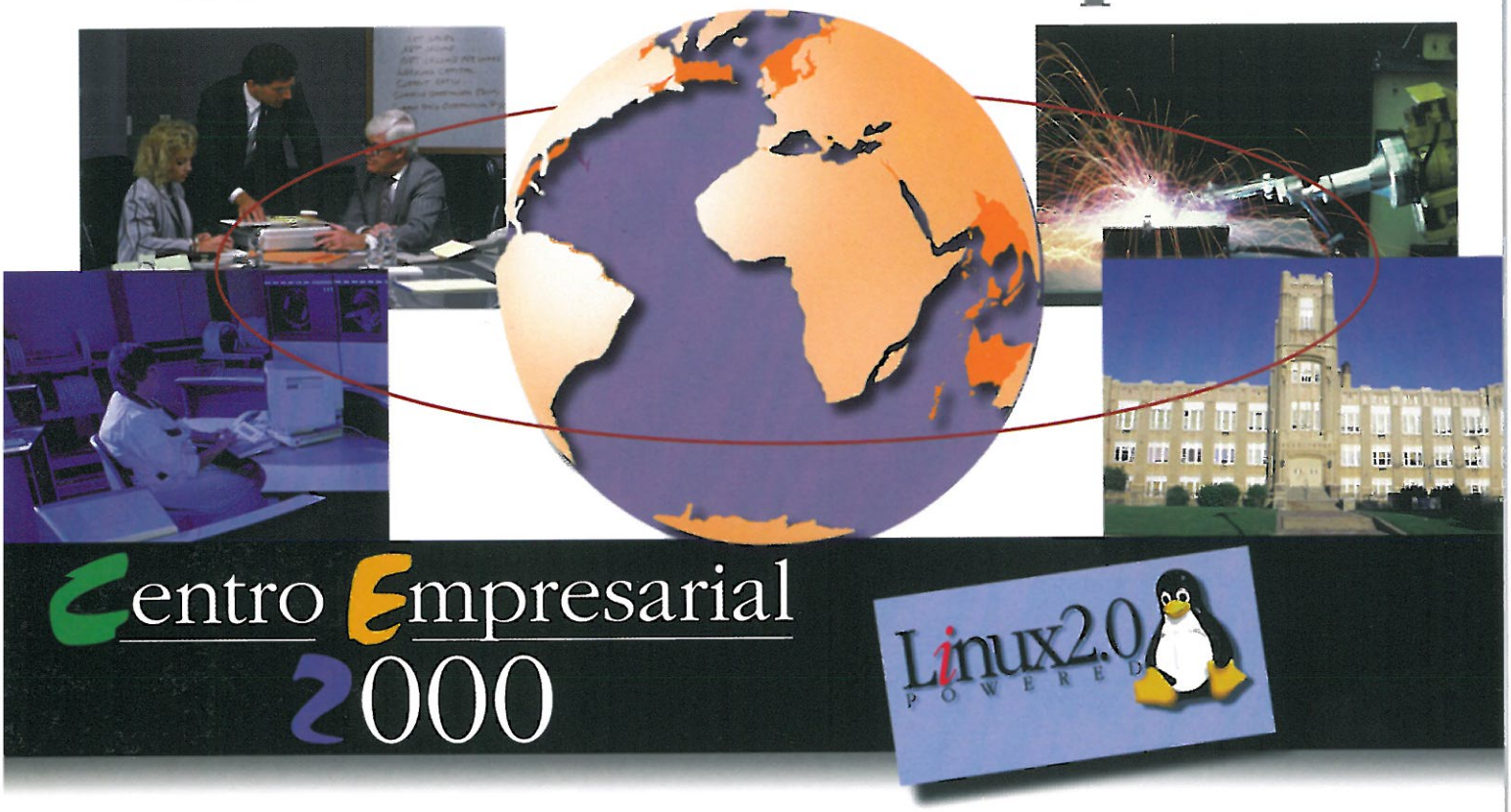
Programación de Threads en Delphi

CONTENIDO CD

- Armed Linux 1.1 beta
- Visual Eiffel Lite 3.2
- WinTeX 2000 v1.0.0.25
- AnyJ 1.3.3
- PureLoad 1.1
- ASP-EDIT 2.05 Build 1
- VoiceAction 2.00
- SecurityManager 99 v1.01
- Gauntlet Firewall 5.50
- Qnet 2000

8 413042 303299 00062

Por fin, disfrute de un servidor de Internet en su empresa



Centro Empresarial 2000

Linux2.0 POWERED

CENTRO EMPRESARIAL 2000, PERMITE ADMINISTRAR, A TRAVÉS DE PÁGINAS WEB, UNA INTRANET DE FORMA CÓMODA, RÁPIDA Y EFICAZ, SIN NECESIDAD DE PROFUNDOS CONOCIMIENTOS INFORMÁTICOS.

COMPATIBLE CON LA MAYORÍA DE LAS REDES ACTUALES.

El sistema incluye:

- SERVIDOR DE PÁGINAS WEB
- SERVIDOR DE CORREO ELECTRÓNICO
- SERVIDOR DE FTP
- Nº DE ACCESOS SIN LÍMITE, TANTO PARA LOS USUARIOS DE SU RED, COMO PARA LOS QUE SE CONECTEN EN REMOTO (DESDE SU DOMICILIO, DESDE UN PORTÁTIL, COLABORADORES,...)
- Nº DE BUZONES SIN LÍMITE
- CREACIÓN Y MANTENIMIENTO DE LISTAS DE CORREO
- ALOJAMIENTO DE DOMINIO Y DIRECCIÓN IP FIJA
- 25 MB DE ESPACIO EN NUESTROS SERVIDORES, SI NO QUIERE ESTAR CONECTADO 24 H. A INTERNET
- ESTADÍSTICAS DETALLADAS DE ACCESOS DE CADA CUENTA, LUGARES VISITADOS,... PARA UN CONTROL TOTAL DE SUS USUARIOS
- INSTALACIÓN DEL SISTEMA OPERATIVO LINUX Y DEL CENTRO EMPRESARIAL 2000
- MANTENIMIENTO REMOTO DEL SISTEMA Y HOT LINE
- SISTEMA PROGRAMABLE DE LLAMADAS AUTOMÁTICAS
- HASTA 100 MB DE TRANSMISIÓN MENSUAL

Coste Total: 175.000 ptas.

- OPCIONES ADICIONALES: SERVIDOR DE FAX, CONEXIÓN ENTRE DIVERSAS SEDES, FORMULARIOS ELECTRÓNICOS, CURSOS DE FORMACIÓN, ACCESO POR NODO LOCAL, ETC...

Buscamos Distribuidores

Virtual SOFTWARE

www.virtualsw.es

Cartagena 52
28028 Madrid
Tel: 91 355 76 67
Fax: 91 355 28 95

SÓLO PROGRAMADORES

Número 62
SÓLO PROGRAMADORES
es una publicación de
REVISTAS PROFESIONALES S.L.

Editor
Agustín G. Buelta
Director
Javier Amado Buiza
Coordinador Técnico
Eduardo De Riquer Frutos
Coordinadoras de Redacción
Gema Romero Moreno-Manzanaro
Cristina Peña del Pozo
Colaboradores
Constantino Sánchez, Juan Luis Ceada, Jorge Delgado, Javier Sanz, Adolfo Aladro, Enrique de la Lastra, Jordi Agost, Vicente A. Sánchez Werner
Maquetación y Tratamiento de Imagen
J. Santos, F. Risco
Consultas técnicas
atecnica@virtualsw.es

Publicidad
Paloma Seidel
Tel.: (91) 304 78 46
Mariano Sánchez (Barcelona)
Tel.: (93) 322 12 38
Pepín Gallardo (Barcelona)
Tel.: 617 09 36 68

Suscripciones
Rosa Tabares
Tel. (91) 304 87 64 Fax: (91) 327 13 03

Preimpresión
Grebe
Impresión
I.G.Pantone
Distribución
Motorpress Ibérica

La revista Sólo Programadores no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados.
El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 29-2-2000
Precio en Canarias, Ceuta y Melilla:
938 ptas. sin I.V.A.
Con sobretasa aérea: 975 ptas. sin I.V.A.

EDITORIAL

Programando en multitarea

Aquellos programadores que trabajan y crean aplicaciones basadas en estructuras Cliente/Servidor tendrán conocimiento sobre los **Treads**. Estos hilos permiten trabajar bajo sistemas multitareas de un modo mucho más ágil que en sistemas convencionales.

La programación con **Treads** se está imponiendo, y no sólo en grandes empresas con enormes sistemas e infraestructuras, sino que existen *Pymes* con servidores *NT* que necesitan aprovechar al máximo las capacidades de su servidor. Para ello no está de más disponer de los conocimientos necesarios en programación, para trabajar con estos hilos y más en concreto con *Delphi*. Por ello, en este número dedicamos nuestro artículo de portada a este tema.

Del mismo modo, tampoco podíamos dejar escapar las novedades que nos ofrece *Delphi* 5 dedicándole un pequeño, pero instructivo, reportaje para explicarlas y ver en qué nos puede beneficiar dependiendo del uso que cada uno suela hacer de él.

En la *Suite* de *Internet* que estamos construyendo hablaremos sobre uno de los aspectos que cada día utilizan más usuarios y que permite conversar desde cualquier parte del mundo utilizando el teclado de nuestro *PC*. Hablamos, como no, del *chat*.

También continuamos con las series de artículos que muchos de vosotros (por las cartas recibidas) esperábais con impaciencia como: *API* de Telefonía (II), La tecnología *ASP* (III), *DNS* a fondo (III), Criptografía (III), Transferencia de archivos punto a punto (V), etc...

Como no podía ser menos, en el mes de noviembre ha acontecido una de las ferias más importantes del año: COMDEX'99, en Las Vegas. Allí hemos estado y te contamos todo lo que en ella ha acontecido, tanto las novedades como las nuevas tendencias, o las conferencias de los gurús de la informática.

Además de las secciones a las que estáis acostumbrados y que son de última hora, como: la ayuda al lector, las noticias y las novedades editoriales.

En el próximo número volveremos puntualmente con más material, que esperamos os guste.

Javier Amado
DIRECTOR

SUMARIO 62

SÓLO PROGRAMADORES

6 Noticias

Las principales novedades en el mundo de la programación aparecen en esta sección tan imprescindible. Dedicada a todos aquellos que no quieren dejar de estar al día en el mundo de la programación.

10 Contenido del CD-Rom

Como siempre en el *CD* que acompaña a la revista podréis encontrar los listados de todos los artículos, junto con los mejores programas y las actualizaciones más novedosas. Este mes queremos destacaros: *Visual Eiffel Lite* 3.2, *WinEdit* 2000, *VoiceAction* 2.00 y *LanExplorer* 3.5.

22 Telecomunicaciones

TAPI: API DE TELEFONÍA (II)

Después de ver en el número anterior las principales funciones de telefonía, en esta ocasión profundizaremos un poco más en los detalles y comenzaremos el proyecto en *Visual Basic* para utilizar *TAPI* mediante llamadas a la *DLL TAPI32*.

30 Multimedia

PROGRAMAR UN REPRODUCTOR MULTIMEDIA CON DELPHI (II)

Continuamos en esta entrega el desarrollo de nuestro reproductor. Como prometimos aprenderemos algunos trucos que pueden resultar de gran utilidad en futuros proyectos, aprovechando la facilidad que posee *Delphi,* para usar cualquier función de la *API* de *Windows*.

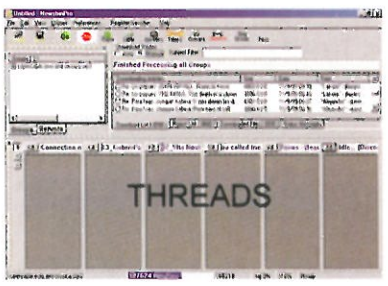
40 COMDEX Las Vegas '99

EL PC HA MUERTO, VIVA EL PC

Como todos los meses de noviembre se ha celebrado en Las Vegas la feria informática más importante del mundo, *COMDEX*, que en su vigésimo aniversario ha marcado el punto de partida de lo que podríamos llamar la era *post-PC* o quizás mejor la era *Internet*. Nosotros estuvimos allí y en estas páginas os comentamos sus principales conclusiones.

12 Portada Delphi

PROGRAMACIÓN DE THREADS (I)

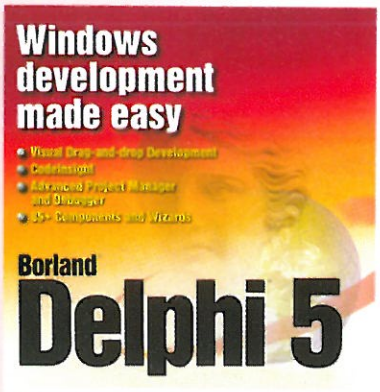


Iniciamos en este número una nueva serie sobre la programación de Threads. En esta primera entrega abordaremos los aspectos relacionados con su creación y tratamiento, y dejaremos para después aspectos más avanzados. Acompañaremos todas las explicaciones teóricas con pequeños ejemplos, para que de esta forma, se afiancen los conocimientos que vayamos adquiriendo.

36 Herramientas

LAS NOVEDADES DE DELPHI 5

Como ya os anunciamos la nueva versión de *Delphi* ya está en la calle. En este artículo, y de una forma rápida, os explicamos las novedades más interesantes que nos trae la versión 5 de esta popular herramienta de *Borland*, empresa que ahora se conoce bajo el nombre de *Inprise*, en diversos apartados tales como bases de datos o *Internet*.



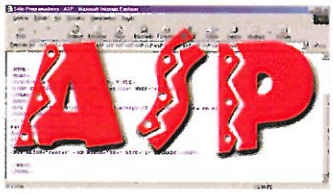
44 Linux

DNS A FONDO (III): SERVICIOS DE DNS CON BIND 8.2.X.

Después de haber sentado las bases teóricas del *DNS* y de ver el tipo de información que puede contener, en este artículo explicaremos cómo implantar un servidor *DNS* en nuestra máquina desempeñando diversas funciones útiles: caché de *DNS*, servidor *DNS* de una red local o cualquier otra aplicación que nos interese.

52 Internet

LA TECNOLOGÍA ASP (III): FORMULARIOS Y MÁS FORMULARIOS



Hasta el momento hemos utilizado formularios en los diversos ejemplos que hemos desarrollado, y más o menos se ha comenzado a ver la forma de manejarlos con páginas *ASP*. Ahora es el momento de profundizar en todos estos aspectos, de forma que seamos capaces de realizar aplicaciones *Web* avanzadas.

58 Java

CRIPTOGRAFÍA (III)

Los algoritmos de clave privada han sido ampliamente utilizados e implementados en diferentes lenguajes, por eso tanto el usuario como el sistema en el que se van a aplicar los prefieren. En este artículo mostramos un último conjunto de algoritmos de clave privada, entre los que se incluyen *IDEA* y *Blowfish*, detallándose además los tipos de modos de operación en bloque disponibles para éstos.

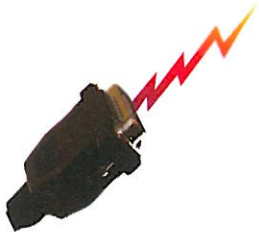
66 Visual Basic

SUITE DE INTERNET (IV): CREACIÓN DE UN CHAT

¿Cómo podemos añadir una comunicación de tipo *chat* a nuestros programas? ¿Cuáles son las aplicaciones que puede tener dicha comunicación sobre la *suite* que estamos desarrollando? ¿Cuáles son los pasos que debemos seguir? ¿Cómo lo podemos realizar con el menor número de controles posibles? A éstas y otras preguntas intentaremos responder a través de este artículo.

72 Redes

TRANSFERENCIA DE ARCHIVOS PUNTO A PUNTO (V)



El diseño de un protocolo de comunicaciones parte de la definición de unas pautas de funcionamiento y termina con la implementación de esas pautas en un código que las cumpla. Después de haber definido y delimitado esas pautas y sus posibles excepciones, en este artículo y en el próximo terminaremos de programar el código de un componente *Delphi* que las desarrolle, para un protocolo de nivel de enlace.

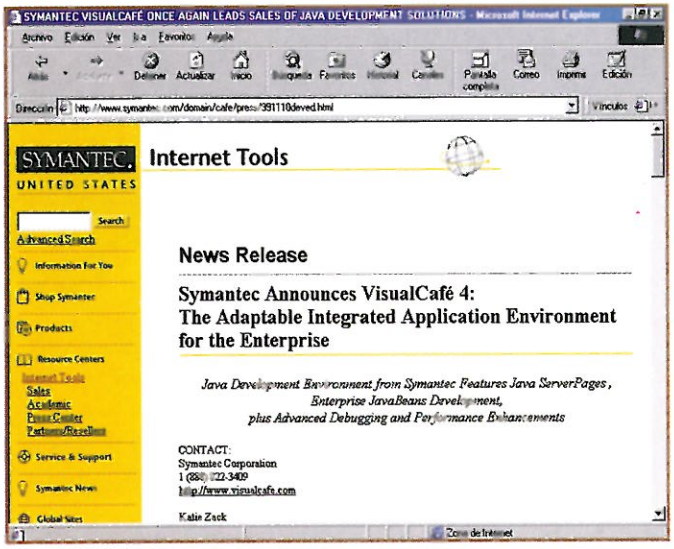
78 Libros

Las últimas novedades editoriales sobre programación y los libros que nos han parecido más interesantes aparecen en estas páginas, con una sencilla reseña que os ayude a conocerlos y a decidir cuál es el que más os conviene.

80 Dudas técnicas

Como siempre en esta sección contestamos a todas vuestras dudas. No os preocupéis si os parecen demasiado complejas o demasiado sencillas, siempre podréis acudir a nuestra dirección de e-mail: **solop@virtualsw.es**. Nosotros intentaremos solucionar vuestros problemas.

VISUAL CAFÉ PARA JAVA 4.0 SALE AL MERCADO



Symatec acaba de anunciar la aparición de *Visual Café* 4.0 para *Java*. La nueva generación de esta herramienta ofrece un número importante de ventajas respecto a sus antecesoras, aprovechando la nueva plataforma de *Sun Mycrosystems Java* 2, que incluye soporte para *JavaBeans* (EJB), *Java Server Pages (JSP), Servlets, CORBA,* y otras tecnonologías avanzadas de *Java*.

Un destacable número de empresas está utilizando el lenguaje *Java* en sus negocios, y la creciente necesidad de servidores en este campo ha conducido a la determinación del desarrollo de la creación de una herramienta con las características de *Visual Café* 4.0.

El soporte de *JavaBeans* ha sido un elemento decisivo en el éxito de esta nueva versión.

Esta herramienta está disponible en el mercado en tres versiones diferentes adaptables según las necesidades de los usuarios: estándar, de empresa y para desarrolladores expertos, ofreciendo un rápido y potente entorno de trabajo.

Para más información: **www.symantec.com.**

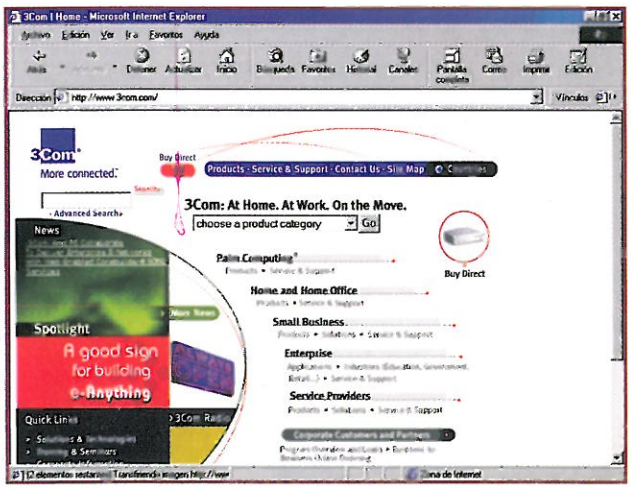
3COM REDISEÑA HARDWARE PARA ACABAR CON LAS ESPERAS DE LA WEB

Teniendo el liderazgo de varias soluciones en el sector de las redes locales *3Com* añadirá nuevas características a sus herramientas que ayudarán a mejorar los sitios *Web* de comercio electrónico a nivel de velocidad y fiabilidad.

Para cumplir este nuevo objetivo que se marca esta empresa, *3Com* contará con la colaboración de *F5 Networks*, una alianza que llega en medio de un "boom" en *Wall Street* por cualquier *hardware* o *software* que funcione en la supresión de "cuellos de botella" en la trasmisión de datos a través de la Red.

Este trabajo conjunto también supone alguno de los primeros pasos en la última estrategia de *3Com* que consiste en enfocar todos sus esfuerzos en redes *backend*, logrando así un nuevo posicionamiento en el mercado en productos como *routers* y *switches* para *Internet*.

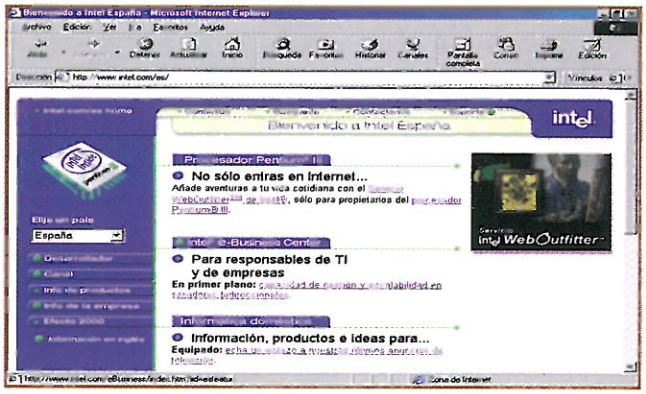
3Com planea revender la tecnología de su familia de *switches, CorelBuilder* 9000. El año próximo esta compañía incorporará la nueva tecnología directamente a sus *switches*.



Para más información: **www.3com.com**

INTEL Y NORTEL NETWORKS PERMITIRÁN CREAR REDES DE NUEVA GENERACIÓN

Intel Corporation y *Nortel Networks* anuncian sus planes de trabajar juntas para permitir crear una nueva generación de dispositivos de red programables por *software*. Estos planes requieren que *Nortel Networks* añada elementos de su *software Open IP Enviromment* a la arquitectura *Internet Exchange (IX)* de *Intel*. La combinación proporcionará a los desarrolladores una completa plataforma abierta para construir equipos de red avanzados basados en la aquitectura *IX*.



Este esfuerzo conjunto se centrará en tres componentes clave: la interoperavilidad, un marketing conjunto y *kit* de desarrolladores. Ambas compañías prevén trabajar juntas para permitir que el procesador de red *IXP1200* de *Level One* de *Intel*, opere con elementos del *software Open IP Enviromment* de *Nortel Networks*.

La arquitectura *IX* de *Intel* es una nueva estructura para diseñar equipos de redes y telecomunicaciones utilizando silicio programable. Consiste en una cartera de componentes procedentes de la filial de *Intel Level One Communications*, incluyendo su buque insignia: el procesador de red *IXP1200*.

El *software Open IP Environment* constituye un *software* de encaminamiento e *IP* de nueva categoría que permite, basándose en *Internet*, una gran variedad de servidores, ordenadores personales, dispositivos de movilidad, *set-top boxes* y procesadores.

Para más información visitar las páginas: **www.intel.com,** y **www.nortelnetworks.com.**

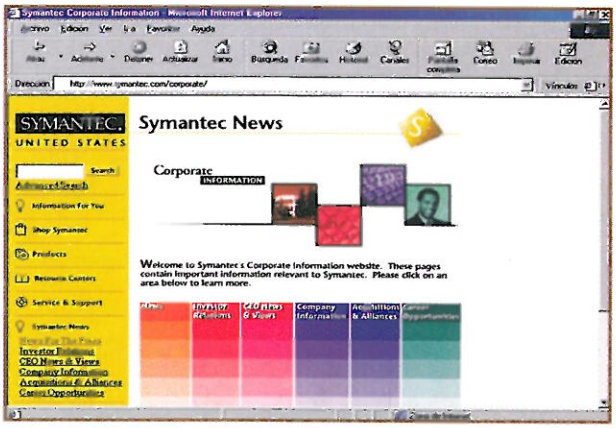
SYMANTEC DESCUBRE LA VACUNA CONTRA EL VIRUS W32.FUNLOVE.4049

La compañía *Symantec* ha anunciado el descubrimiento de la primera vacuna contra el virus *W32.FunLove.4099* por parte de sus investigadores. Este virus utiliza una nueva estrategia para residir en memoria dentro del sistema *Windows NT* con lo que encontrar una cura supuso todo un desafío ya que los antivirus tradicionales no conseguían ningún efecto sobre él.

W32.FunLove.4099 ataca a los archivos de seguridad de *Windows NT* modificando su *kernel*, de este modo tiene la capacidad de transformar su sistema de seguridad. El virus crea también un programa para replicarse automáticamente en segundo plano sin que el usuario note pérdida alguna en el rendimiento del sistema.

Para proteger de inmediato el sistema ante este nuevo virus, los usuarios pueden descargarse las últimas definiciones de virus haciendo uso de la utilidad *LiveUpdate* o directamente desde la página *Web*: **www.symantec.com/avcenter/download.htm.**

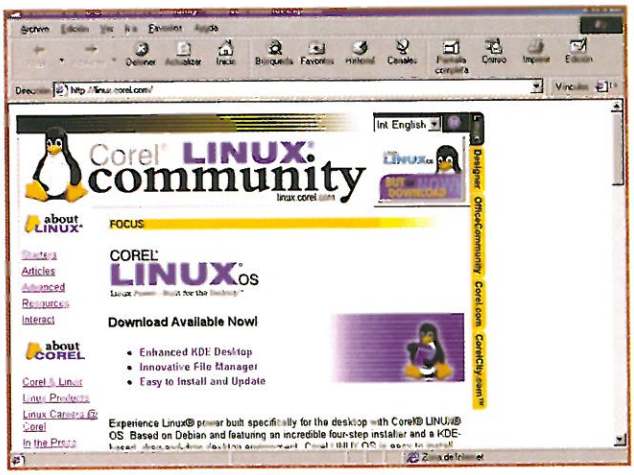
Para más información: **www.Synmatec.com**



EL SISTEMA OPERATIVO COREL LINUX YA ESTÁ DISPONIBLE

La compañía *Corel Corporation* ha anunciado que el esperado sistema operativo *Corel Linux* está ya a disposición de los usuarios para ser descargado gratuitamente.

Existen tres versiones diferentes de este sistema operativo, la versión de descarga gratuita, que se puede encontrar en el sitio *Web* de *Corel*: **www.corel.com;** la versión estándar, y la versión *Deluxe*. Los usuarios que deseen la versión gratuita en un *CD-ROM* pueden adquirirla llamando al servicio de atención al cliente de *Corel*.



La versión de descarga incluye el sistema operativo *Corel Linux* (basado en *Debian*, con el núcleo *Linux* 2.2), el escritorio *KDE* mejorado (basado en *Qt*), *Corel Install Express, Corel Update, Corel File Manager, y Netscape Communicator*.

Por su parte la versión estándar incluye tres *CD-ROM*, el de la instalación principal, el de circulación libre que contiene aquellos componentes de la instalación principal que pueden copiarse libremente sin necesidad de adquirir licencias adicionales; y un tercero con el código abierto de circulación libre.

El tercer tipo consiste en la versión *Deluxe*, que incluye todos los componentes de la versión estándar más dos *CD-ROM*, ambos con aplicaciones registradas que no se pueden copiar.

El sistema operativo *Corel Linux* es un importante paso adelante en la creación de un gran mercado *PC* para *Linux*. Esta innovadora versión se considera la más sencilla de instalar hasta la fecha, según miembros de la propia compañía.

Para más información: **www.corel.com.**

MSN.ES LANZA COMMUNITIES

El servicio Communities de MSN estará disponible en la dirección **http://communities.msn.es/** a partir del 29 de Noviembre. Dicho servicio *on-line* permite al usuario crear fácilmente páginas *Web*. *Communities* facilita esta labor gracias a un amplio abanico de herramientas, incluyendo páginas *HTML* personalizadas, álbumes de fotos, tablones de anuncios, salas de *chat,* etc, con lo que los diferentes usuarios pueden comunicarse en un site administrado por ellos mismos, cubriendo las necesidades de cada uno de ellos.

Así se podrá crear de modo sencillo y gratuito comunidades *Internet* sobre cualquier tema imaginable o navegar por una lista de comunidades ya creadas.

Las principales características de este sistema son: la creación de páginas de comunidades, para ayudar a los miembros a organizar y guardar la información; las páginas de inicio personal, con la que cada usuario puede crear distintivos visuales personalizando su página de inicio por medio de fotos, *clip art* o *links* sin la necesidad de tener conocimientos de *HTML*. La gestión personalizada de correo electrónico, álbum de fotos, etc.

HERRAMIENTAS DE PROGRAMACIÓN

ENTORNOS DE PROGRAMACÓN

ARMED LINUX 1.1 BETA

Distribución del sistema operativo *Linux* para *Windows*, destinado a usuarios que lo utilizan por primera vez o para todos aquellos que quieran utilizar *Linux* como un sistema operativo secundario. Soporta aplicaciones compatibles con *Red Hat Package Manager*.

EDITCNC 2.06

Editor de texto diseñado para la programación y edición *CNC (Computer Numerical Control)*. Ofrece las herramientas habituales de los editores de texto avanzados. El tamaño del fichero únicamente está limitado por la capacidad del ordenador.

PERL SCRIPTING TOOL 1.0.6

Completo entorno de desarrollo *Perl*. Incluye una utilidad interna *HTM2PERL*, una librería de subrutinas, un verificador de sintaxis, un asistente de código, accesos directos definibles por el usuario, etc.

VISUAL EIFFEL LITE 3.2

Completo sistema de desarrollo para *Eiffel*. Consta de un visor, un depurador y un compilador que produce el código de 32 *bits* para equipados con procesador *INTEL*. Entre otras posibilidades, ofrece acceso directo al *API* de *Windows* y generación de aplicaciones *GUI*.

WINEDIT 2000

Potente editor de textos que incorpora un cliente *FTP*. Incluye un lenguaje de macros con más de 550 funciones y plantillas para *C*, *C++*, *BASIC*, ensamblador *x86* y otros. Ofrece soporte para *Java JDK* y para páginas *HTML*, un visor *HTML* y un generador de *tags HTML*.

WINTEX 2000 V1.0.0.25

Editor de texto especialmente optimizado para su utilización con ficheros *TeX*. Ofrece un sistema de diseño de páginas creado para la impresión de trabajos científicos y matemáticos. Incluye un editor de tablas, barras de herramientas con comandos utilizados frecuentemente y numerosas plantillas de código.

YET ANOTHER APPLICATION FRAMEWORK 1.0A8 BETA

Entorno de trabajo de aplicaciones *C++*, que simplifica la creación de aplicaciones para *Macintosh*, *Microsoft Windows* y *X-Windows*. Incluye una librería, código fuente, utilidades, ejemplos, un editor de los diálogos y *Rcompile*.

LENGUAJES

VISUAL BASIC

DV IDE TOOLS LITE 1.0

Pequeño programa que mejora el *IDE* de *Visual Basic* permitiendo automatizar las tareas repetitivas. Puede localizar cadenas de código, visualizar una lista de procedimientos, renombrar controles, insertar líneas de comentarios y mostrar estadísticas del proyecto.

SLICE AND DICE FOR VISUAL BASIC 1.8

Herramienta de automatización de tareas de programación específicamente diseñada para *Visual Basic*. Permite crear, modificar, borrar y generar formularios, módulos de código, módulos de clase, controles de usuario, ejecutables *ActiveX*, librerías, ficheros, directorios, bases de datos, etc.

URTHMAN'S MDB CODE GENERATOR 1.00.0191

Analiza una base de datos de *Microsoft Access* y genera un módulo de código fuente en *Visual Basic* para grabar y escribir registros de la base de datos. Permite seleccionar o deseleccionar cualquier tabla del fichero para su inclusión en el código generado.

JAVA

ANYJ 1.3.3

Entorno de desarrollo *Java*. Ayuda a analizar y comprender librerías de clases complejas para mejorar la productividad en la escritura, reutilización y mantenimiento del código fuente.

JAVALOG 0.7

Colección de clases *Java* que dan formato y escriben eventos para uno o más dispositivos *LOG*. Agiliza el desarrollo de dispositivos, ahorrando tiempo de codificación. Además permite al usuario filtrar de modo dinámico los eventos para adecuarlos a sus necesidades.

PURELOAD 1.1

Herramienta de análisis y prueba del rendimiento de aplicaciones servidor. Simula el trabajo de cientos de usuarios realizando diferentes tareas y genera estadísticas en tiempo real. La información puede

ser utilizada para analizar los "cuellos de botella" del sistema, el rendimiento y parámetros de calidad.

HTML

ACCESS2WEB 1.2

Convierte los ficheros de bases de datos de *Access* a formato *HTML*. Para ello hay que coger el código *HTML,* copiarlo a *Access2Web,* abrir la base de datos de *Access* e insertar sus campos en el código *HTML;* el programa genera todos los ficheros *HTML,* uno para los registros de la base de datos y un fichero índice.

ASP-EDIT 2.05 BUILD 1

Editor de páginas *ASP*. Facilita la inclusión de expresiones y funciones, como variables del servidor, ordenes *VBScript,* expresiones *ASP,* etc. Cuenta con una herramienta para cambiar la configuración del servidor *ASP* en el Registro de *Windows,* asistentes para crear expresiones *SQL* y ficheros *GLOBAL.ASA.*

MERACL IMAGEMAP GENERATOR 3.3.1

Genera mapas de imágenes para páginas *HTML*. El procedimiento es muy sencillo ya que únicamente es necesario seleccionar un rectángulo, un círculo o un polígono y arrastrarlo hasta una imagen. Después se incorpora el enlace, el *frame* correspondiente y el texto.

OTROS

3DPIECHART 3.1

Componente *ActiveX* para diseñar diagramas circulares *3D* interactivos. Genera en tiempo real gráficos *3D* y permite modificar el color, la textura y los gradientes, crear fondos utilizando texturas e imágenes en mosaico, establecer el tamaño y ángulo de los fragmentos de los diagramas, etc.

AXLABELPLUS 1.01

Control *ActiveX* que soporta todas las propiedades de etiquetas estándar. Incluye funciones para controlar la posición, definir hiperenlaces, establecer de forma independiente el alto y el ancho, cambiar la apariencia del cursor cuando pasa sobre el control, etc.

DUCK REPORT 2.0C

Colección de herramientas de creación de informes para *Delphi*. Incluye un visor, un asistente de informes y librerías de componentes para informes y códigos de barras. Ofrece soporte para múltiples fuentes de datos, comandos *SQL,* previsualizar e imprimir los trabajos y utilizar gráficos.

DYNAPLOT 1.1

Control *ActiveX* de gráficos para aplicaciones técnicas y científicas. Cambia de forma dinámica los diferentes elementos de los gráficos como el ancho de bordes y líneas, las fuentes, los cursores, etc. Incluye un modo *bitmap* para la reproducción exacta de *bitmaps* de datos.

VOICEACTION 2.00

Control *ActiveX* de reconocimiento de voz. Proporciona un conjunto de elementos ya preparados para añadir a aplicaciones de reconocimiento de voz. Facilita la creación de una base de datos de vocabulario y permite al programador conseguir una mayor exactitud en el reconocimiento de las palabras.

HERRAMIENTAS Y UTILIDADES

ZZ'S LOGIN MANAGER 1.1.0

Guarda nombres de usuario y claves de acceso en una base de datos de seguridad. Puede encriptar y desencriptar automáticamente los nombres y *passwords,* generar nuevas claves, realizar búsquedas, etc.

CODE MAGICIAN 2.0.0

Sistema de seguimiento de las versiones de nuestro *software*. Permite conocer los detalles de las versiones de *software* que tienen nuestros clientes como el número de versión, los nuevos ficheros que necesitan, comentarios sobre cada versión, los problemas que presentan y las soluciones encontradas, etc.

DATA MANAGER 2.1

Procesa y manipula datos de manera lógica y sencilla. Lee y escribe ficheros delimitados permitiendo especificar el delimitador. Facilita la labor de escoger el modo en que vamos a procesar los datos y transformarlos en otros formularios.

EXAMDIFF PRO 2.2

Herramienta de comparación de ficheros de texto que muestra línea a línea las diferencias existentes entre los mismos. Incluye distintas opciones de visualización y ofrece la posibilidad de comparar en ocasiones diferentes las mismas áreas de texto.

SECURITYMANAGER 99 V1.01

Encripta rápidamente ficheros individuales y carpetas completas utilizando el método *DES (Data Encryption Standard)*. Los ficheros originales son inutilizados de modo que es imposible volver a recuperar la información con las utilidades estándar de recuperación de ficheros.

TASKINFO2000 V2.01

Muestra información de todos los procesos en ejecución en equipos que trabajen bajo *Windows 95/98*. Ofrece una amplia información sobre cada proceso y sobre el sistema. Además, permite ejecutar nuevos procesos, cambiar su prioridad y finalizar procesos en ejecución.

REDES

AUTONET 3.3
Crea y automatiza un *router* de marcación automática en una red. Puede funcionar en equipos *Windows, Mac, Unix* o *Amiga* y conectar cualquier ordenador de una red a una *ISP*. Trabaja como un *firewall* y soporta todos los protocolos más populares *(FTP, HTTP, RealAudio,* etcétera).

GAUNTLET FIREWALL 5.50
Firewall seguro para *Windows NT,* en las conexiones a *Internet, intranets* y *extranet.* Emplea el método más seguro de protección *firewall,* una aplicación *gateway.* Ofrece gestión integrada, potente encriptación para redes privadas y seguridad de contenidos.

LANEXPLORER 3.5
Analizador de protocolos de redes y tráfico. *LanExplorer* intercepta los paquetes de información de dentro y fuera de una estación de trabajo o del servidor. Soporta numerosos protocolos como *TCP/IP, 802.3, VLAN, Apple, Novell* y *Microsoft.*

QNET 2000
Herramienta de modelización de redes neurales. Entre otras aplicaciones, puede ser usada para resolver problemas de automatización, toma de decisiones, diagnóstico médico e inteligencia artificial.

VISUALROUTE 4.1A
Utilidad que analiza el estado de las comunicaciones en *Internet procesandp* todas las conexiones *IP* en paralelo. Identifica problemas de conectividad y muestra los resultados de forma gráfica sobre un mapa mundial.

DOCUMENTACIÓN/ TUTORIALES

ARTICLES FOR PUBLICATION 1.0
Manual electrónico que contiene respuestas a una colección de dudas frecuentes sobre el tema de las publicaciones electrónicas. Además, incluye 400 artículos escritos por profesionales disponibles para ser publicados en una revista electrónica o en un sitio *Web.*

FAT32 AND HARD DISK PARTITIONING GUIDE 1.08
Tutorial sobre las *FAT* y los comandos *FORMAT* y *FDISK.* En un fichero en formato *HELP* se presentan completas explicaciones acerca de la creación de volúmenes *FAT32,* cómo aumentar su rendimiento, el procedimiento de conversión de *FAT32,* etc.

SHAREWARE AUTHOR'S RESOURCE GUIDE
Guía que proporciona a los autores de programas interesante información para el éxito de negocios de *shareware.* Esta guía proporciona datos sobre recursos, contactos importantes, la distribución y el marketing de las aplicaciones.

URTHMAN'S SOCKETS DATABASE 1.00.0031
Base de datos de ayuda que facilita el mantenimiento de ficheros de servicios de red asociados a la configuración de la red *TCP/IP.* Es una referencia funcional de las asignaciones de *sockets TCP/IP.*

VISUAL BASIC AND DATABASES 1.0
Tutorial de introducción del uso de *Visual Basic* para el acceso y mantenimiento de bases de datos para personas que no cuenten con experiencia en bases de datos, pero que sí conozcan *Visual Basic.*

IMPRESCINDIBLES

ANTIVIRUS
AntiViral Toolkit Pro 3.0.129
AVP Virus Enciclopedia
McAfee VirusScan 4.0.3
Panda Antivirus 6.10.00 Platinum

GRÁFICOS
Icon Bank 4.0 Gold Edition Web Edition / Desktop Edition
IconForge 4.5
MicroAngelo 98 v4.77
Paint Shop Pro with Animation Shop 6.0
Reptile 2.0
SureThing CD Labeler 2.0
ThumbsPlus 4.02
Xara WebStyle 1.2

INTERNET
Añadir Pro 4.00.008
AutoWinNet 6.0 Beta 1
Copernic 2000 v4.0
Cuentapasos 3.75
CuteFTP 3.5
Dial-Up Magic 1.8
Eudora Light 3.0.6
GetRight 4.1.1
Go!Zilla 3.5
Guardián 1.1
HomeSite 4.01
MIRC 32 5.61
Net Vampire 3.3
URL Organizer 2.3
WinGate 3.0.5

MULTIMEDIA
AudioCatalyst 2.01
CDH Media Wizard 4.11
COWON Jet-Audio 4.6
Sonique 1.20
WinAMP 2.50e

NAVEGADORES
Internet Explorer 5.0.2314.1003
NeoPlanet 5.0.0.1045
Netscape Communicator 4.7
Opera 3.60

PROGRAMACIÓN
Hackman 4.0b3
Help & Manual 2.25
InstallConstruct 3.2.1
Java Development Kit 1.2.2
UltraEdit Professional Text/HEX Editor 6.20b
Windows Registry Guide 1.3
WinHex 8.75

UTILIDADES
Adobe AcrobatReader 4.0
Advanced Registry Tracer (RegFix) 1.0a
Babylon Translator 2.1
CallCenter 3.5.8
Calendar Builder 3.2j
DirectX 7.0
Emergency Recovery System 8.73
Nero 4.0.7.0
SiSoft Sandra 99.8.5.30
Where Is It? 2.15
Windows Commander 4.01
WinZip 7.0

ATENCIÓN:
En caso de problemas con el CD-ROM envíelo por correo ordinario, a la atención del SERVICIO TÉCNICO DE SÓLO P., incluyendo en el interior del sobre sus datos personales, a la siguiente dirección:
C/ San Sotero, Nº 5, 1º Planta,
28037 (Madrid)

Programación de Threads (I)

Juan Luis Ceada Ramos.
Programador en ARCABE Formación y Servicios Informáticos.
Octavio Martín Díaz.
Profesor de Análisis de la Universidad de Sevilla.

Una de las características que define un sistema operativo se refiere a la posibilidad de ejecutar varios procesos de una forma concurrente, aprovechando al máximo la potencia del procesador y ofreciendo un servicio a varios usuarios. Para aprovechar todas estas ventajas, nada mejor que gestionar los distintos procesos utilizando los *threads* (hilos) y la herramienta *RAD Delphi.*

INTRODUCCIÓN

En esta primera entrega abordaremos los aspectos relacionados con la creación y tratamiento de *threads*. Acompañaremos las explicaciones teóricas con pequeños ejemplos, para que de esta forma, se afiancen todos los conocimientos que vayamos adquiriendo con los diferentes artículos.

Para próximas entregas dejaremos aspectos algo más avanzados, como por ejemplo el acceso a bases de datos desde diferentes hilos, la creación de *threads* usando la *API* de *Windows* (sin usar la clase *Tthread* que proporciona *Delphi),* y algunas cosas más relaciona-

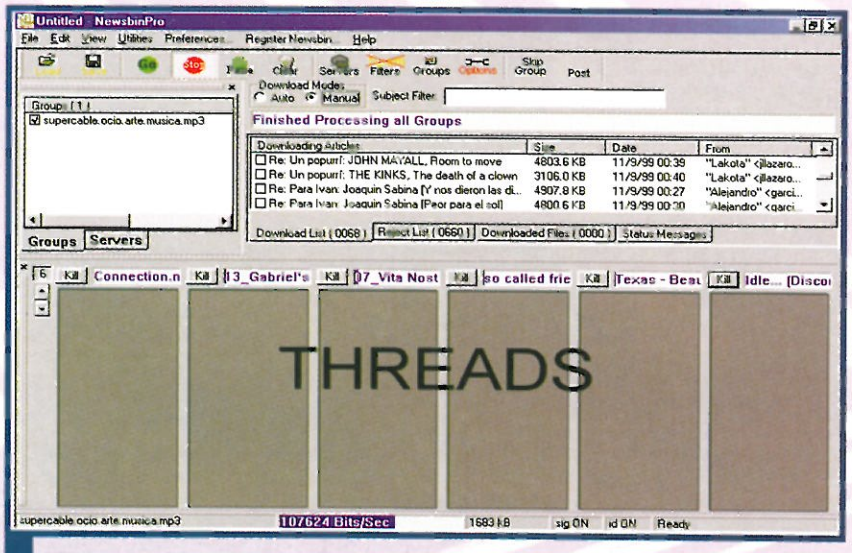


Figura 1.- Ejemplo de una aplicación que usa múltiples hilos de ejecución.

das con el acceso a recursos compartidos desde diferentes *threads*.

¿QUÉ ES UN THREAD?

Una aplicación *Win32* consta de uno o más procesos. A su vez, cada proceso puede estar formado por uno o más hilos de ejecución. Cada proceso tiene su propio espacio de direcciones virtuales, datos y código. Además puede tener asignados determinados recursos del sistema, como ficheros, puertos de *e/s*, etc.

La clase TThread evita recurrir a la API para crear threads

A su vez un proceso puede contener uno o más hilos de ejecución. Todos los hilos de un mismo proceso comparten el mismo espacio de memoria, las variables globales y los recursos que el sistema operativo haya asignado al proceso.

Cada hilo puede realizar una tarea diferente, o bien puede haber varios hilos realizando la misma tarea en paralelo. En pocas palabras, podemos pensar que un hilo no es más que un programa dentro de otro programa.

Podemos considerar a los threads como programas dentro de otros programas

Un sistema operativo multitarea divide el tiempo de procesador disponible entre los *threads* que lo necesiten. En *Windows*, la multitarea es de tipo *preemptive*. Esto significa que el sistema operativo es el encargado de dividir el tiempo de procesador en pequeñas porciones y asignar estas porciones de tiempo a cada una de las tareas que compiten por ocupar el procesador.

Puesto que el tiempo que un *thread* (en condiciones normales) permanece activo es de unos 20 ms, el usuario tiene la sensación de que todas las tareas se están ejecutando a la vez.

En un sistema monoprocesador, la multitarea es simulada. Realmente no se ejecutan más instrucciones aunque se hayan lanzado varios *threads*, puesto que sólo habrá uno en ejecución en cada instante.

Windows NT sí que proporciona multitarea real (en sistemas multiprocesador)

La cosa cambia si el sistema es multiprocesador, puesto que en este caso es posible repartir la carga de *threads* entre los diferentes procesadores. En este caso sí se dispone de multitarea real.

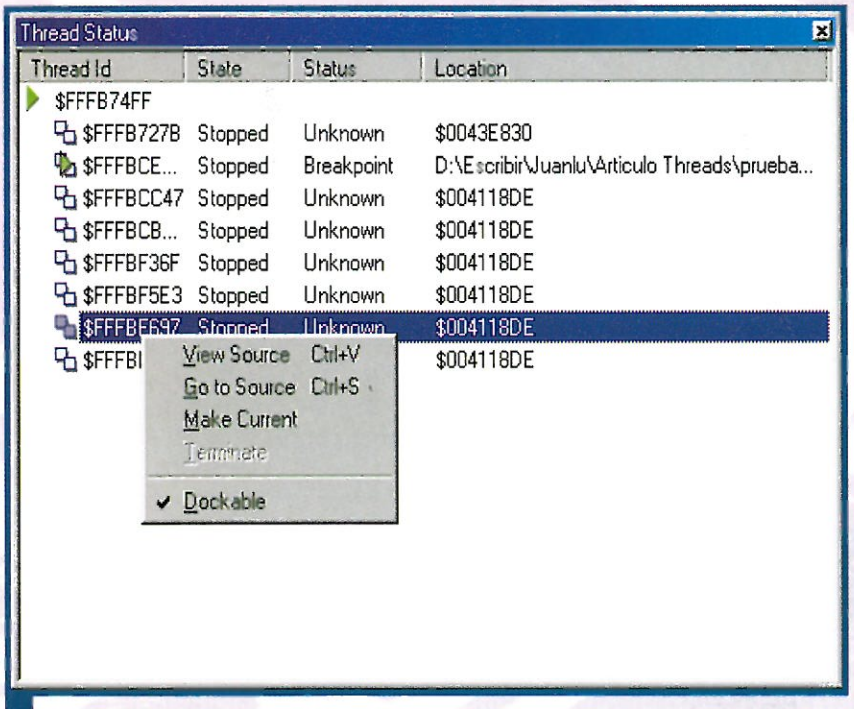


Figura 3.- Ventana de depuración de *threads*.

VENTAJAS E INCONVENIENTES

VENTAJAS

Se reducen los cuellos de botella: En una aplicación *multithread*, es posible crear un *thread* que se encargue de acceder a los recursos (que son las operaciones más lentas), mientras que otros se encargan de realizar los cálculos (evitando, de esta forma, esperas innecesarias). De esta manera se evita esperar a que terminen las operaciones de *E/S*, para efectuar determinados cálculos.

Se mejora el rendimiento: A menudo desarrollamos programas que son inherentemente paralelos (por ejemplo, una rutina de multiplicación de matrices). Es posible dividir un programa en *threads*, de forma que todas se ejecuten a la vez.

Tabla 1. Propiedades y métodos de la clase *TThread*

Nombre	Tipo	Descripción
FreeOnTerminate	Propiedad	Indica si el thread se destruirá automáticamente cuando termine su ejecución.
Priority	Propiedad	Indica la prioridad que tendrá el thread.
ReturnValue	Propiedad	Valor de retorno usado por el método WaitFor. Equivale al Result de las funciones.
Suspended	Propiedad	Indica si un thread se encuentra suspendido.
Terminated	Propiedad	Indica si el thread ha recibido la orden de finalizar su ejecución.
ThreadId	Propiedad	Identifica al thread.
Create	Método	Crea una instancia de la clase TThread. Si recibe como parámetro **False**, el thread comenzará a ejecutarse en cuanto se cree.
Destroy	Método	Destructor de la clase.
Execute	Método	Código que se ejecutará cuando se ponga en marcha el thread.
Resume	Método	Hace que un thread que estaba en modo suspendido siga ejecutándose.
Suspend	Método	Pausa la ejecución del thread activo.
Synchronize	Método	Se ejecuta el método que recibe como parámetro en el hilo principal, evitando, de este modo, los problemas derivados de la programación multi-thread.
Terminate	Método	Indica al thread, poniendo la propiedad Terminated a **True**, que debe finalizar su ejecución.
WaitFor	Método	Espera a que el thread termine y devuelve el valor de la propiedad ReturnValue.

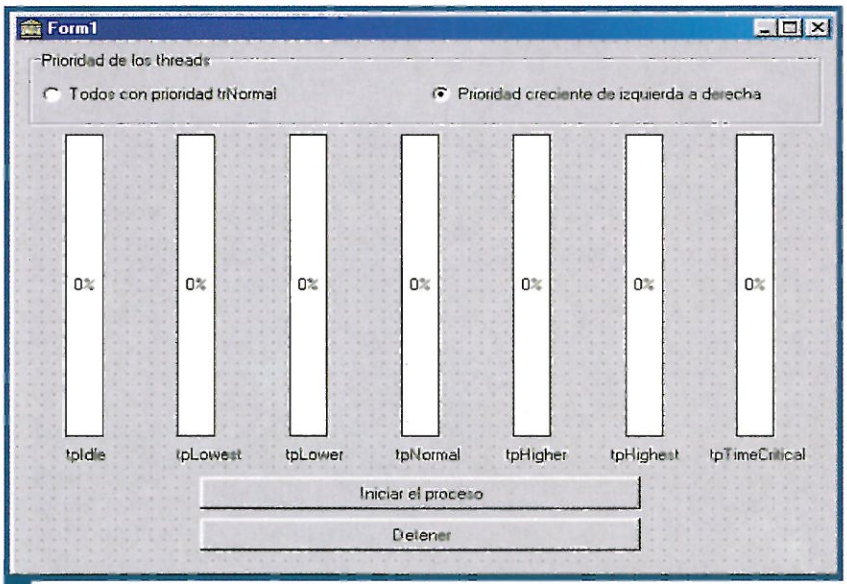


Figura 2.- Formulario principal de la aplicación de ejemplo.

Si el sistema es monoprocesador, es posible que no notemos la diferencia entre usar *threads* o no. Pero si algún día nuestra aplicación se ejecuta en un sistema multiprocesador, notaremos un incremento del rendimiento considerable. Además, gracias a los *thread* tenemos la posibilidad de asignar más tiempo de *CPU* a las tareas más prioritarias.

INCONVENIENTES

La programación de aplicaciones *multithread* suele ser más complicada, puesto que normalmente hay que introducir código de sincronización de procesos. Además, la depuración suele convertirse en un trabajo mucho más pesado.

```
C:\Archivos de programa\Borland\Delphi4\Projects\Unit3.pas
Unit2 | Unit3
uses
  Classes;

type
  TMiThread = class(TThread)
  private
    { Private declarations }
  protected
    procedure Execute; override;
  end;

implementation

{ Important: Methods and properties of objects in VCL can only be used i
  method called using Synchronize, for example,

      Synchronize(UpdateCaption);

  and UpdateCaption could look like,

    procedure TMiThread.UpdateCaption;
    begin
      Form1.Caption := 'Updated in a thread';
    end; }

{ TMiThread }

procedure TMiThread.Execute;
begin
  { Place thread code here }
end;

5: 44 | Modified | Insert
```

Figura 4.- Esqueleto generado por *Delphi* de la clase *Mithread*.

LA CLASE THREAD

En la mayoría de las aplicaciones *multithread* que desarrollemos, usaremos la clase *TThread* (ver Tabla 1) que *Delphi* proporciona para crear los hilos de ejecución. Esta clase encapsula las funciones de creación de hilos correspondientes a la *API* de *Windows*.

En algunos casos, como por ejemplo si necesitamos controlar atributos relacionados con la seguridad, tendremos que recurrir a programar directamente con la citada *API*.

Los threads incrementan el rendimiento de los sistemas multiprocesado

DEPURACIÓN DE APLICACIONES MULTITHREAD

Lo primero que tenemos que hacer es mostrar el cuadro de diálogo *Threads Status* (ver Figura 3), mediante la combinación de teclas **CtrlAlt+T**. Cuando se produce un evento de depuración (por ejemplo, un punto de ruptura, una excepción, etc.), este cuadro de diálogo muestra el estado de cada *thread*.

Haciendo *clic* con el botón derecho del ratón sobre cada uno de los *threads* de la lista, podemos ir al punto del código fuente donde se produce el evento

ICON MEDIALAB BUSCA A
LOS MEJORES PROGRAMADORES

ERES el mejor en lo que haces, pero quieres más. Quieres estar donde innovar sea parte del día a día. Te entusiasmaría colaborar en el futuro de las mejores compañías del mundo.
Quieres llevar los medios digitales un paso más allá.

SOMOS la consultora líder en comunicación digital. Asesoramos a nuestros clientes en el diseño de su estrategia de futuro en internet.
Estamos a la vanguardia en comunicación estratégica y en las tecnologías más avanzadas.
Contamos con los mejores profesionales integrados en una sólida red internacional.

TE ofrecemos la oportunidad de trabajar para las principales compañías en los más avanzados proyectos en internet.

Estar HOY donde los demás estarán en el futuro.

Todo lo que debes hacer es ir a
www.iconmedialab.es/jobs
y contarnos dónde quieres estar en los próximos años.

También puedes enviarnos tu solicitud por correo a:
Icon Medialab España
C/Alcalá, 21-8o Dcha.
28014 Madrid

Icon Medialab tiene oficinas en Estocolmo, Bruselas, San Francisco, Londres, París, Hamburgo, Milán, Madrid, Helsinki, Tampere, Copenhague, Oslo y Kuala Lumpur.

ICON MEDIALAB

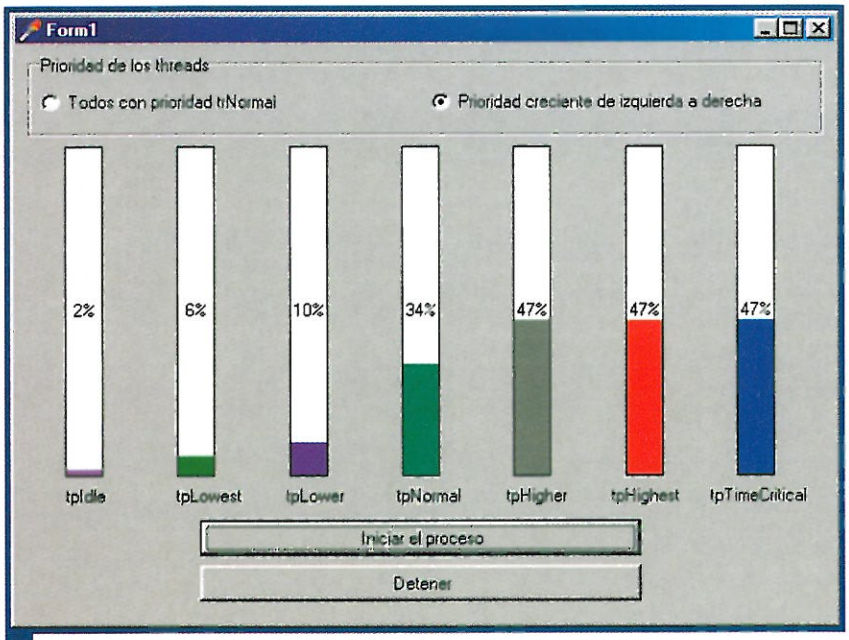


Figura 5.- *Threads* con diferentes prioridades.

de depuración. También podemos hacer que el *thread* seleccionado sea el activo, con lo cual cualquier cosa que hagamos (como por ejemplo pulsar F8), afectará a dicho *thread*.

APLICACIÓN DE EJEMPLO

En el *CD-ROM* que acompaña a la revista se incluye el código de varios ejemplos de aplicaciones *multithread*. Ahora vamos a ir viendo cómo se ha implementado una de ellas. De esta forma compaginaremos teoría con práctica, para que todo sea más fácil de entender.

La programación *multithread* resulta algo más complicada

Empezaremos por crear el formulario principal. En él añadiremos un par de botones, así como siete barras de progreso *(TGauges,* en la pestaña *Samples).*

En la Figura 2 se puede observar cómo se han distribuido los componentes en el formulario principal.

CREACIÓN DEL THREAD

Como ocurre con otras clases, no podemos trabajar directamente con la clase *TThread.* Tendremos que crear una nueva clase descendiente de ella, para lo que usaremos la opción del menú **File->New->Thread** (ver Figura 4).

EL THREAD PRINCIPAL

INICIAR EL THREAD

En este caso, tendremos que cambiar algunas propiedades del *thread,* como su prioridad, antes de que empiece a ejecutarse. Para ello no nos queda más remedio que crear un nuevo constructor para la clase. En otras ocasiones puede que no necesitemos cambiar el comportamiento del *thread,* por lo que no sería necesario crear un nuevo constructor.

En la sección de implementación incluiremos el código de inicialización de la clase. Nuestro método **Create** (ver Figura 8), recibe como parámetros el componente *TGauge* que debe

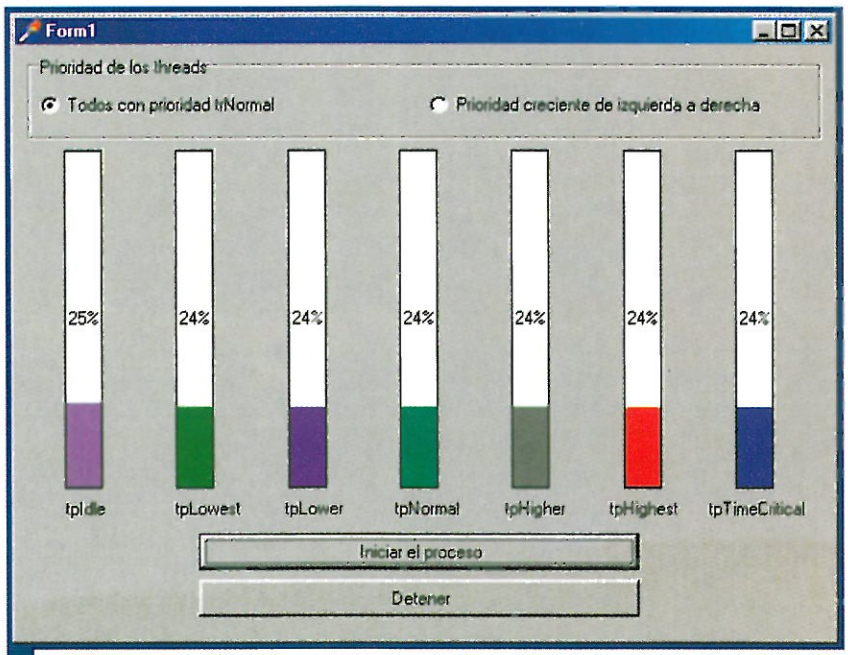


Figura 6.- *Threads* con la misma prioridad.

actualizar y la prioridad con la que se debe lanzar el *thread*. Copiaremos los parámetros a variables privadas del *thread*.

Para variar el comportamiento por defecto de un thread hay que crear un nuevo constructor

Además, llamaremos al constructor *Create* de la clase padre con el parámetro *CreateSuspend* con el valor **True**. De esta forma el *thread* no empezará a ejecutarse hasta que llamemos al método *Resume*, lo que permite que antes de comenzar su ejecución, podamos cambiar su prioridad.

Si hubiésemos llamado al constructor *Create* con el parámetro *CreateSuspend* a **False**, la ejecución del *thread* habría dado comienzo nada más finalizar el proceso de creación (con lo que no hubiésemos podido cambiar su prioridad).

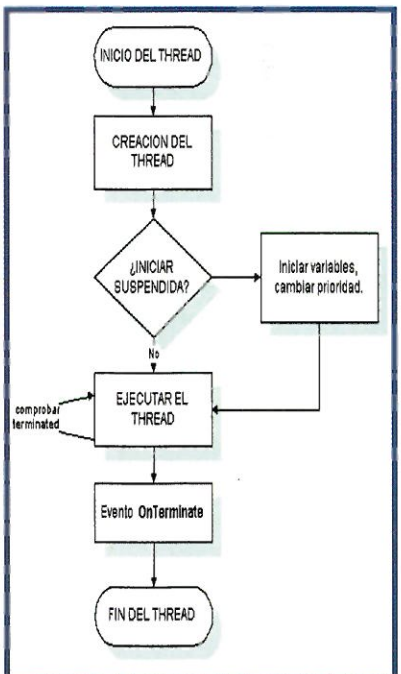


Figura 7.- Ciclo de vida de un *thread*.

```
Unit2.pas
Unit1 | Unit2

procedure TMiTh1.ActualizarBarras;
begin
     FBarra.Progress:=Contador;
end;

constructor TMiTh1.Create(Barra:TGauge; Prioridad:TThreadPri
begin
     FBarra:=Barra;
     inherited Create(True);
     OnTerminate:=LiberarRecursos;
     Priority:=Prioridad;
     Resume;
end;
procedure TMiTh1.LiberarRecursos;
begin
     FBarra:=Nil;
end;

procedure TMiTh1.Execute;
begin
     contador:=1;
     while (contador<=100000) and not (terminated) do
     begin
          Inc(contador);
          Synchronize(ActualizarBarras);
     end;
end;

end.

50: 1  Modified  Insert
```

Figura 8.- Método *Execute* de la aplicación de ejemplo

La propiedad *Priority* permite asignar la prioridad que tendrá el *thread*. Puede valer: **tpIdle, tpLowest, tpLower, tpNormal, tpHigher, tpHighest** y **tpTimeCritical**. Los nombres de las constantes son autodescriptivos, por lo que sólo comentaremos algo sobre el primer valor.

Para cambiar la prioridad es necesario crear el thread en modo suspendido

Si la prioridad de un *thread* es **tpIdle**, *Windows* no interrumpirá la ejecución de otros hilos para pasar a ejecutar un hilo que tenga esta prioridad. Los hilos de este tipo sólo se ejecutan cuando el procesador esté ocioso (por ejemplo, cuando se está esperando a que el usuario introduzca algún dato por teclado).

En la Figura 5 hay siete *threads* en ejecución. Cada uno de ellos tiene una prioridad diferente. Se han lanzado todos a la vez, pero avanzan con distintas velocidades (aunque todos ejecutan el mismo código).

Es posible comunicar threads usando variables globales

En la Figura 6 tenemos otra captura del mismo ejemplo, pero en este caso todos los *threads* tienen prioridad de valor **tpNormal**, por lo que todas las barras avanzan a la vez.

EL MÉTODO EXECUTE

Después de llamar al método *Resume*, comienza la ejecución del código contenido en el método *Execute*. Se puede pensar en este método como en un programa externo que lanza nues-

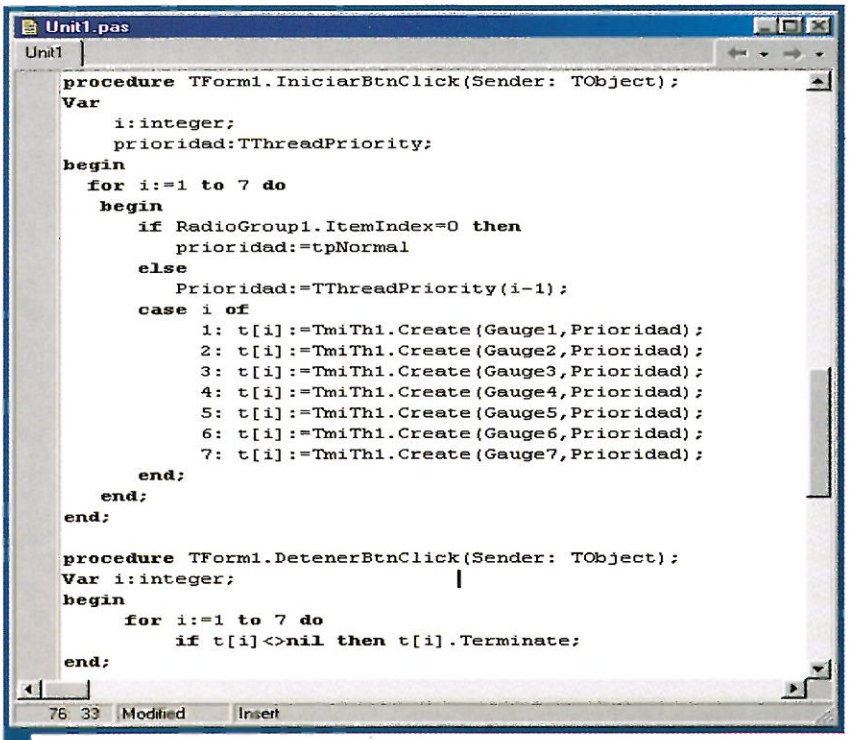


Figura 9.- Código de los botones **Iniciar** y **Detener** del formulario principal.

tra aplicación, sólo que comparte el mismo espacio de memoria que la aplicación principal.

Precisamente esto es lo que va a provocar mayores problemas a la hora de trabajar con *threads*, puesto que habrá que tener muchísimo cuidado a la hora de acceder a la memoria, así como a los componentes *VCL* de los formularios.

Aunque por otro lado, el hecho de que todos los hilos compartan el mismo espacio de memoria, permite que éstos se comuniquen de una forma muy sencilla, usando variables globales. En este caso, y como se puede ver en la Figura 8, nos limitamos a incrementar un contador. El valor de este contador se refleja en el *TGauge* asociado al *thread*.

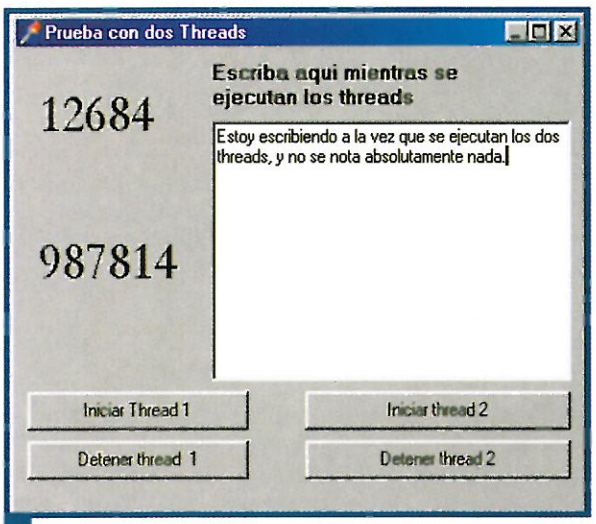


Figura 10.- Ejemplo 2, versión que usa *threads*, en ejecución.

Cuando se usan objetos de la VCL, hay que tener en cuenta que no se garantiza que sus propiedades y métodos puedan ser usados dentro de un *thread* sin que surjan problemas. Al acceder a las propiedades o métodos de un componente es posible que se realicen operaciones en memoria que interfieran con operaciones que se ejecutan en otros *threads*.

Sólo desde el hilo principal podremos acceder al VCL

Es por esto que cualquier operación que implique acceder a componentes *VCL* se debe realizar en el hilo principal de la aplicación. Para ello se define una rutina que realice las acciones requeridas.

Todos los *threads* accederán a la *VCL* ejecutando dicha rutina, usando para ello el método *Synchronize*. En la Figura 8 hay un ejemplo , en el se utiliza una rutina que actualiza el *TGauge*.

Un thread puede indicar a cualquier otro que se detenga

Synchronize espera a que el hilo principal entre en el bucle de mensajes y después ejecuta el método que se le pasa como parámetro. Por cierto, la rutina que accede al *VCL*, en nuestro caso *ActualizarBarras*, no puede recibir ningún parámetro.

Puesto que el método *Synchronize* usa el bucle de mensajes, no funciona en aplicaciones de consola. En este caso, es necesario usar otros mecanismos, como por ejemplo, las secciones críticas, los semáforos, etc. Volveremos a este punto en próximas entregas.

No todos los objetos de la librería *VCL* necesitan depender del hilo principal para poder trabajar con ellos. Son los siguientes:

- Componentes de acceso a datos: son seguros siempre que cada *thread* disponga de su propio com-

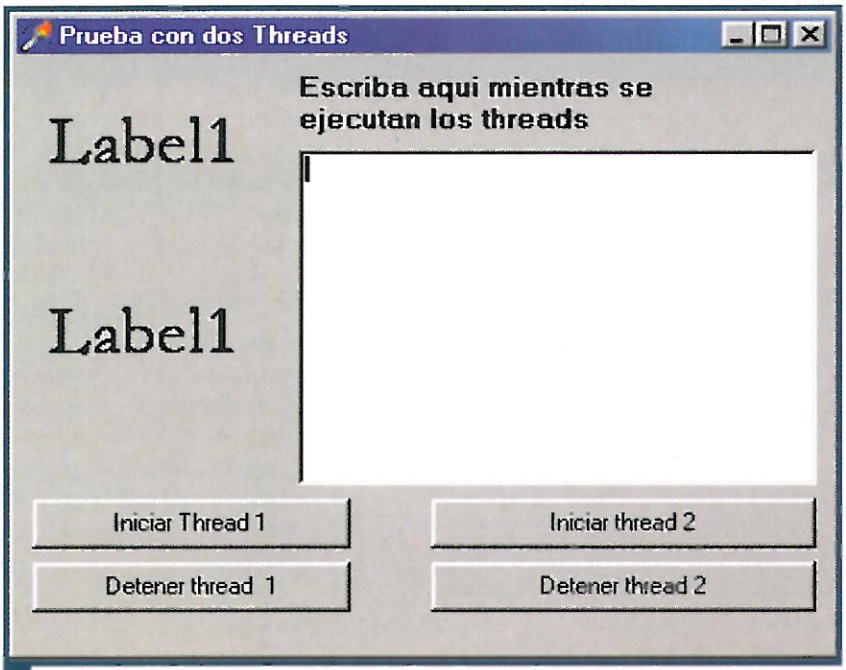


Figura 11.-Ejemplo 2, versión que usa *threads* sin ejecutar.

ponente *TSession*. En realidad hay que tomar precauciones a la hora de acceder a las propiedades de los componentes. Lo veremos con más detalle en próximas entregas.

- Objetos gráficos, como *TFont, TPen,* etc.: La clase *TCanvas* dispone de un método que permite bloquear la superficie de dibujo, por lo que no es necesario usar el método *Synchronize.*

- La clase *TList,* o más exactamente, uno de sus derivados, *TThreadList.*

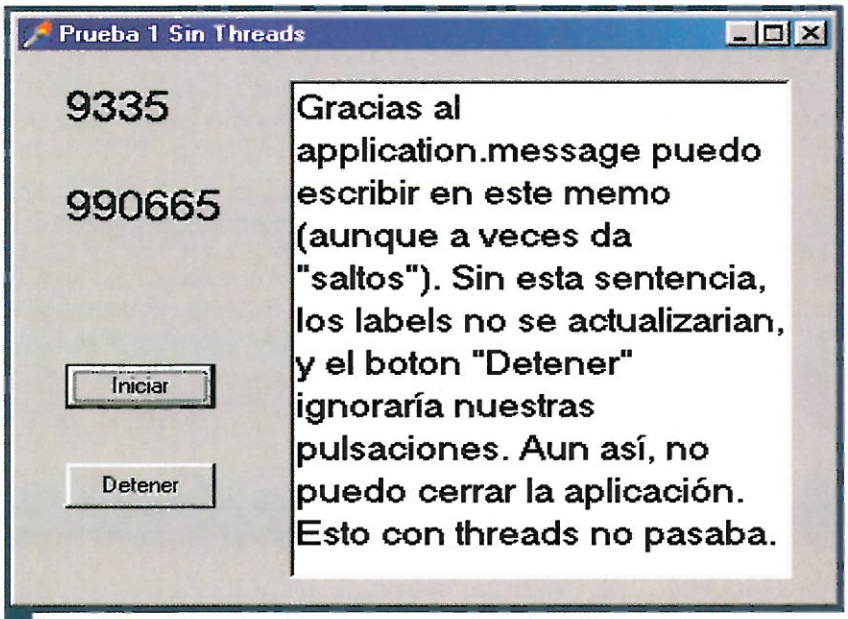


Figura 12.- Ejemplo 2, versión que NO usa *threads.*

FIN DEL THREAD

Un thread comienza a ejecutarse cuando se llama al método *Execute,* y continúa hasta que éste finaliza (ver Figuras 7 y 8). Es decir, el *thread* se lanza, realiza una serie de acciones, y finaliza. Pero a veces puede ser necesario ejecutar estas acciones hasta que se cumpla una determinada condición.

No conviene lanzar muchos threads para no sobrecargar la máquina

Es posible hacer que un *thread* envíe una señal a otro *thread* indicándole que debe finalizar su ejecución. Para ello se usa el método *Terminate*. Este método cambia el valor de la propiedad *Terminated* a **True**. Dentro del método *Execute* debe existir algo parecido a esto:

```
Procedure MiTh.Execute
Begin
While not Terminated do
 RealizarProceso;
End;
```

Es decir, si en nuestro método *Execute* no comprobamos periódicamente el valor de la propiedad *Terminated,* el método *Terminate* no servirá de nada (ver Figura 8). En la Figura 9 se muestra el código del botón **Detener**, que es el encargado de ejecutar los métodos *Terminate* de cada *thread*.

LIBERAR THREADS

Lo más normal es que un *thread* se ejecute una única vez. En este caso, para liberarlo, basta con asignar *True* a

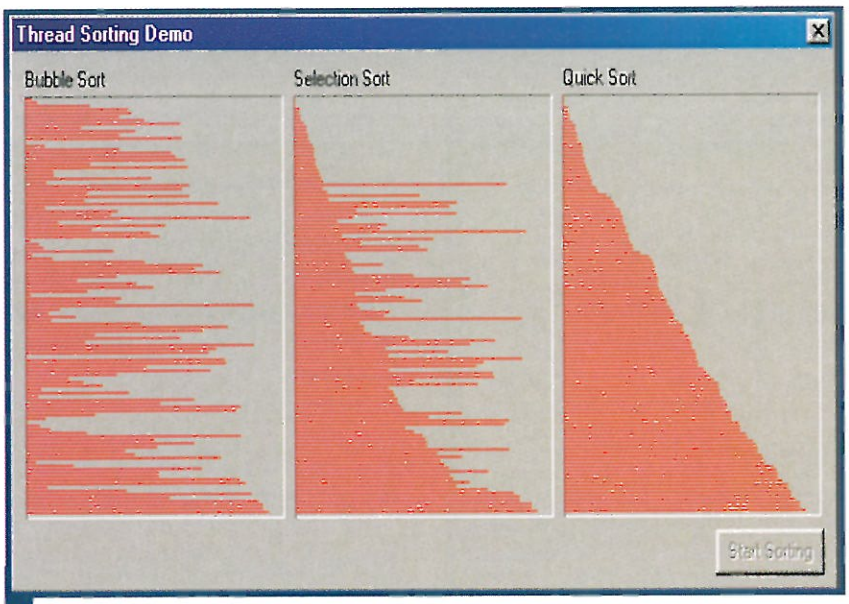


Figura 13.- Aplicación *multitgread* que viene incluida en *Delphi* como ejemplo.

la propiedad *FreeOnTerminate*. De esta forma, cuando finalice la ejecución del método *Execute,* el *thread* se destruirá automáticamente. Este es el comportamiento por defecto de los *threads,* por lo que en nuestro ejemplo no tendremos que hacer nada.

Por otro lado, a veces es necesario ejecutar una determinada tarea muchas veces, en respuesta a un evento externo, como por ejemplo la recepción de un mensaje proveniente de un periférico. En este caso, es posible mejorar la eficiencia de la aplicación implementando una caché para *threads*. También se abordará este tema en próximas entregas.

LIBERANDO RECURSOS

Cuando finaliza la ejecución de un *thread* se produce el evento *OnTerminate*. Es dentro de este evento donde incluiremos todo el código necesario para liberar recursos y dejar el sistema en el mismo estado en el que estaba antes de comenzar la ejecución del *thread.*

El evento OnTerminate salta cuando finaliza la ejecución del thread

Este método no se ejecuta como parte del *thread,* sino que lo hace dentro del contexto del hilo principal, por lo que es posible acceder a cualquier componente o clase de la *VCL* de una forma totalmente segura.

En nuestro caso, nos limitamos a liberar la variable **FGauge**. *(FGauge* = **nil**)

LANZAR LOS THREADS

Una vez que se ha implementado un *thread,* introduciendo código en su método *Execute,* ya es posible lanzarlo. Lo único que hay que hacer es crearlo.

No es conveniente crear muchos *threads*. La sobrecarga provocada por el aumento del número de *threads* puede repercutir en el rendimiento de la aplicación. El límite recomendado, para sistemas monoprocesador es de **16**, si no son demasiado "activos". En caso contrario, se recomienda crear un número inferior.

Es posible crear múltiples instancias de un mismo *thread* para ejecutar código paralelo. Un ejemplo de esto lo tenemos en los servidores de *Internet,* donde por cada cliente se lanza un *thread* que atenderá todas sus peticiones, mientras el servidor continua a la espera de nuevos clientes.

OTROS EJEMPLOS

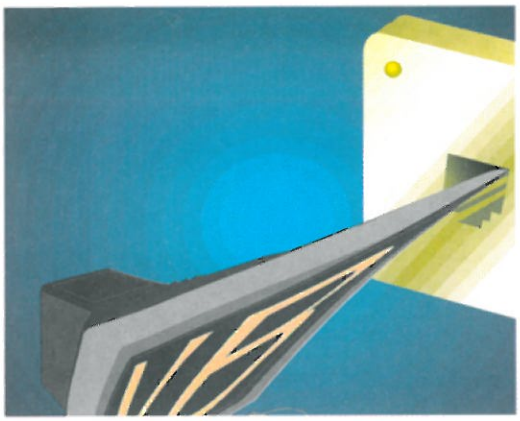
Esto es precisamente lo que hacemos en el ejemplo. Disponemos de una única clase *thread* que se lanza siete veces. Cada *thread* lanzado tiene una determinada prioridad y actualiza el control *TGauge* que se le indica. El código que hace todo esto esta asociado al botón **Iniciar** y se muestra en la Figura 9.

En las Figuras 10, 11, 12 y 13 se pueden observar otros ejemplos que han sido desarrollados para esta serie de artículos. El ejemplo 2 (Figuras 10 y 11) muestra la misma aplicación, en versión *multithread* y *monothread.*

Observe el código fuente (incluido en el *CD-ROM)* intente comprender como funciona, verá cómo no es di-fícil.

En la Figura 12 se muestra el ejemplo que incluye *Delphi* sobre *threads*. Conviene echarle un vistazo a dicho ejemplo, puesto que es muy instructivo. Además de usar *threads,* hace uso de los mecanismos de herencia, clases abstractas, etc. En definitiva, un buen ejemplo, que no se debería pasar por alto.

TODOS LOS MESES EN TU QUIOSCO



TAPI: API de telefonía (II)

Constantino Sánchez Ballesteros.
Técnico Superior de Desarrollo de Aplicaciones

Este mes comenzaremos el proyecto en *Visual Basic* para utilizar *TAPI* mediante llamadas a la *DLL TAPI32*, además de conocer más detalles sobre las funciones de telefonía que vimos en el número anterior.

EL PROVEEDOR DE SERVICIOS MEDIOS (MSP)

Un *MSP* permite a una aplicación un control considerable sobre el medio de un mecanismo de transporte en particular. Además, siempre existe en conjunción con un *TSP*.

Un *MSP* realiza el control de medios implementando interfaces de control de cadena de medios especiales. Expone a *TAPI* los objetos terminal que controlan los puntos finales de esas cadenas de medios.

Como ya se ha dicho, un *MSP* expone objetos terminal especiales a *TAPI*. Con *Windows* 2000 se han instalado *MSP's* que gestionan el protocolo *H323* para telefonía *IP* y otro que gestiona conferencia *multicast* sobre una red con *routers* activados.

En la Figura 1 se puede apreciar la arquitectura de un *MSP*.

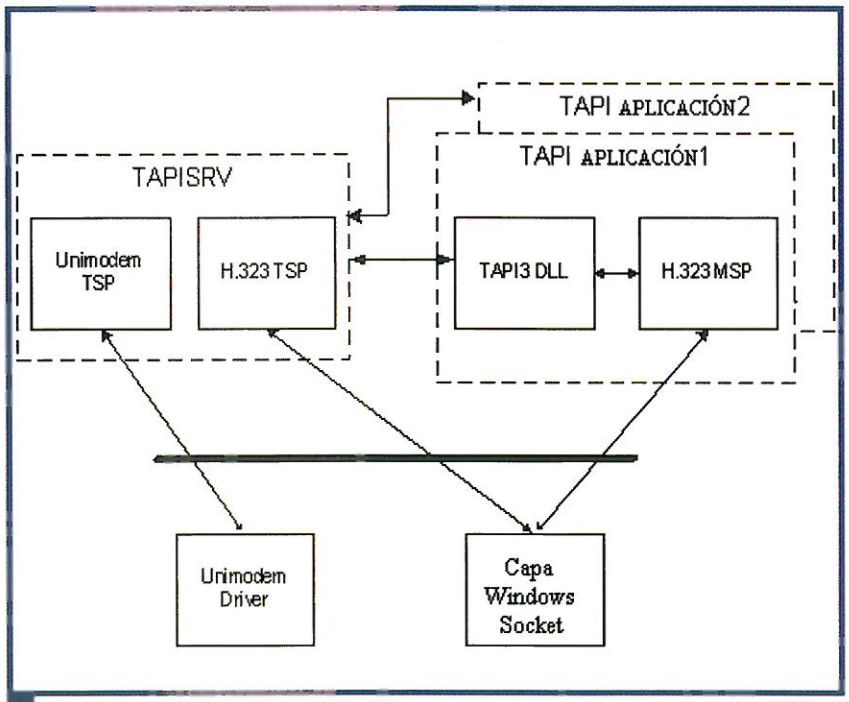


Figura 1.- Arquitectura de un *MSP*.

TIPOS DE TSP'S

En la primera entrega ya vimos lo que era un *TSP* (proveedor de servicios de telefonía). Ahora es el momento de saber qué *TSP's* podemos utilizar.

H323

El *H323 TSP* (*H323.tsp*), cuando es utilizado en conjunción con su

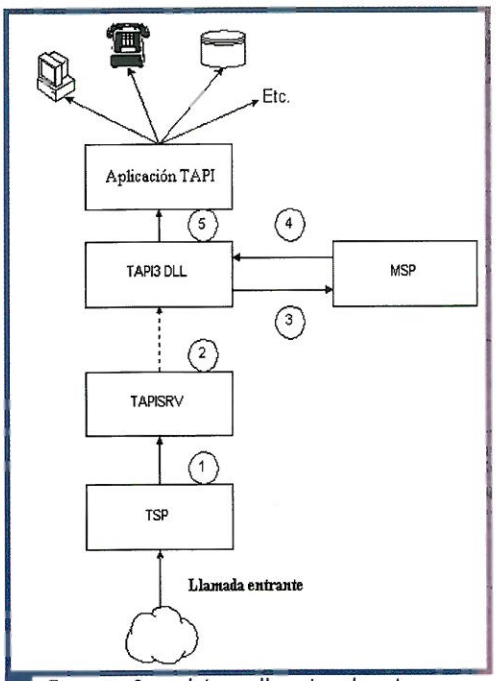


Figura 2.- Control de una llamada utilizando conjuntamente *TSP/MSP*.

MSP (*H323msp.dll*), permite a una aplicación crear y recibir llamadas de audio y vídeo utilizando el protocolo *H.323* para telefonía *IP*.

IPCONF

Este *TSP* (*Conf.tsp*), cuando es utilizado en conjunción con su *MSP* (*Confmsp.dll*), permite acceso a conferencias sobre *Internet*, *Intranets*, etcétera.

NDIS PROXY

Es un componente en modo *kernel* que expone minipuertos *NDIS* como si fueran líneas *TAPI*. Este *TSP* (*Ndptsp.tsp*) permite a *TAPI* comunicarse con el *proxy NDIS*.

UNIMODEM

El *Universal Modem Driver*, *TSP* (*Unimdm.tsp*), permite acceso a la mayoría de *Modems*. Las versiones de *Windows* 95, *Windows* 98, y *Windows* 2000 soportan *voice modems* que permiten cadenas *half-duplex*. El *Unimodem* sobre *Windows NT* 4.0 no soporta *voice modems*.

KERNEL MODE DEVICE DRIVER TSP

El *TSP* en modo *kernel* permite gestionar minipuertos *WAN* (por ejemplo, *NDIS* 4.0) como dispositivos *TAPI*.

ESPECIFICAR MEDIA MODES

La habilidad de una aplicación para tratar las llamadas entrantes o ser el objetivo de llamadas en una línea telefónica viene determinada por el valor utilizado para el parámetro *dwMediaModes* de la función *lineOpen*. Con esta función, la aplicación indica su interés en supervisar o recibir las llamadas entrantes de uno o más *Media Modes* específicos, tal y como se describe en los siguientes casos:

- Una llamada entrante de un cierto *Media Mode* es dada a una aplicación que haya abierto el dispositivo de línea para ese *Media Mode* en particular. Una aplicación puede especificar múltiples *flags* simultáneamente para gestionar múltiples *Media Modes*.
- Una aplicación que quiere manejar llamadas para las que el *Media Mode* actual todavía no haya sido determinado, activaría un medio desconocido para utilizarlo como parámetro de *lineOpen*.
- Una aplicación que quiera manejar llamadas de cualquier *Media Mode* debería indicar esta característica activando todos los *Media Bits* que estén soportados por la línea (estos *bits* pueden obtenerse de *LINEDEVCAPS*).

Todas las aplicaciones que hayan abierto un dispositivo de línea en cualquier modo recibirán una notificación relacionada con ciertos estados generales y eventos que ocurran en el dispositivo

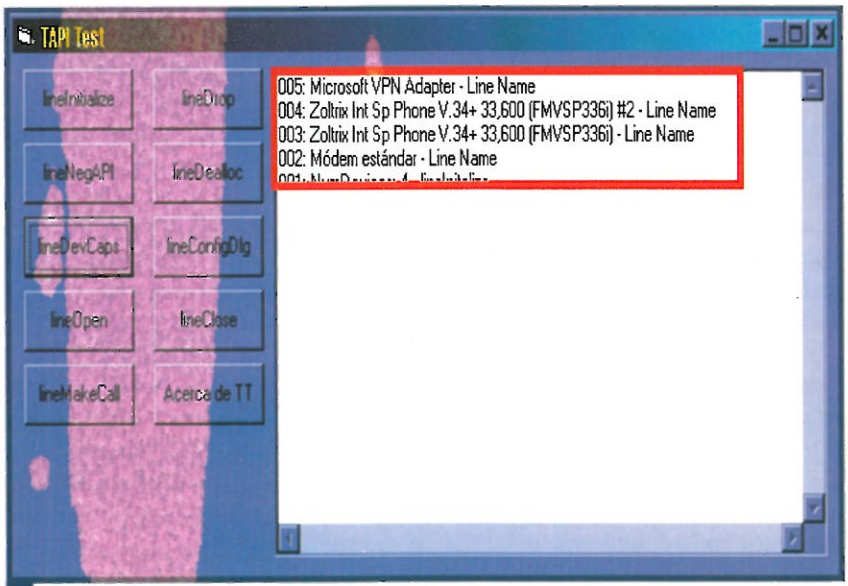


Figura 3.- Resultados obtenidos por *LineDevCaps*.

de línea. Éstos incluyen la línea que está fuera de servicio, la que está disponible, la que está bajo mantenimiento y operaciones de apertura o cierre que estén ocurriendo sobre ella.

La aplicación que no necesite permanecer pendiente de la llegada de un determinado mensaje puede utilizar la función *LineSetStatusMessages* para filtrarlo. La mayoría de los mensajes de estado son desactivados por defecto, de tal forma que la aplicación necesitaría llamar a *LineSetStatusMessages* para volverlos a activar.

La función *LineMonitorMedia* modifica la máscara que controla los mensajes de tipo *LINE_MONITORMEDIA*, pero no afecta a la detección de medios activada por defecto sobre las nuevas llamadas entrantes. Es necesario para cualquier aplicación utilizar la función *lineMonitorMedia*. Esto permite establecer la monitorización de medios sobre cada nueva llamada en la que estemos interesados.

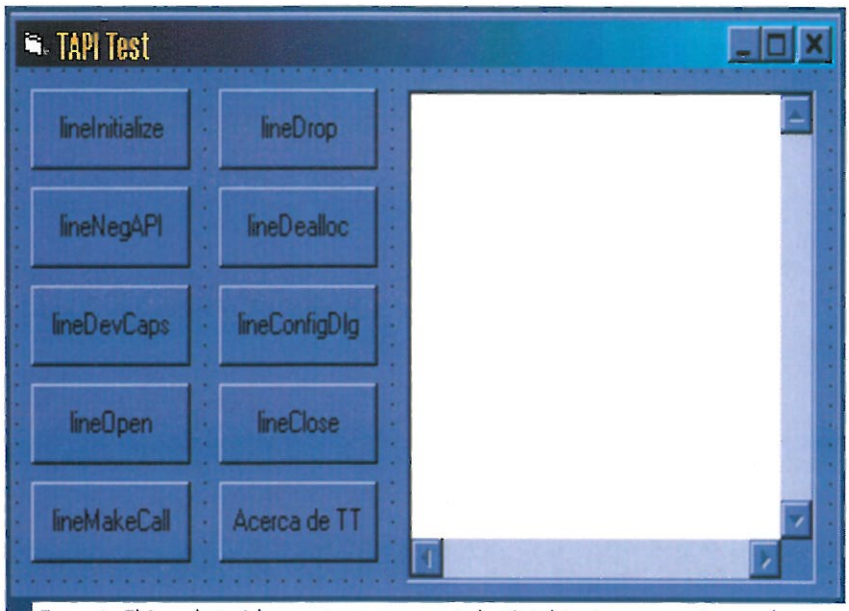


Figura 4.- El formulario del proyecto no posee controles de telefonía, pues accedemos a la interfaz *TAPI* mediante una *DLL*.

PRIVILEGIOS DE UNA LLAMADA

Además del *Media Mode*, una aplicación puede especificar los privilegios de llamada que quiera para las llamadas suministradas. Con los privilegios, la aplicación especifica cuándo quiere monitorizarlas o poseerlas. Para una llamada entrante, sólo se selecciona una aplicación como poseedora, aunque todas las demás aplicaciones con monitorización de llamadas recibirán notificaciones sobre los pormenores de la entrante. Los privilegios típicos que suele especificar una aplicación se detallan en los siguientes puntos:

Los privilegios determinan qué queremos hacer con las llamadas

- Si la aplicación sólo quiere colocar llamadas salientes, se deberá especificar el *flag LINECALLPRIVILEGE_NONE*. La aplicación que no esté seleccionada no será informada de forma automática sobre las llamadas entrantes. Sin embargo, también podemos darnos cuenta de llamadas en la línea con la ayuda de los mensajes *LINE_ADDRESSSTATE* y *LINE_LINEDEVSTATE*. En este punto podremos llamar a la función *LineGetNewCalls* para obtener nuevas llamadas.

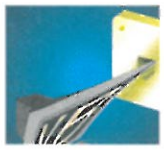
- Si la aplicación sólo quiere monitorizar llamadas, se debe especificar el *flag LINECALLPRIVILEGE_MONITOR*. Además, la aplicación también recibirá notificaciones sobre las llamadas salientes colocadas por otras aplicaciones.

- Si la aplicación quiere aceptar llamadas entrantes de un *Media Mode* concreto (o varios), se debe especificar el *flag LINECALLPRIVILEGE_OWNER* y uno o más *flags* del tipo *LINEMEDIAMODE_*, siempre dependiendo de lo que queramos hacer.

- Si la aplicación quiere controlar llamadas no clasificadas (llamadas entrantes de un *Media Mode* desconocido), deberemos especificar *LINECALLPRIVILEGE_OWNER* y *LINEMEDIAMODE_UNKNOWN*.

- En otros casos, la aplicación debería especificar el *Media Mode* que está interesada en gestionar y establecer el parámetro *dwPrivilege* como *LINECALLPRIVILEGE_OWNER*.

Una aplicación que quiera informarse de todas las llamadas de la



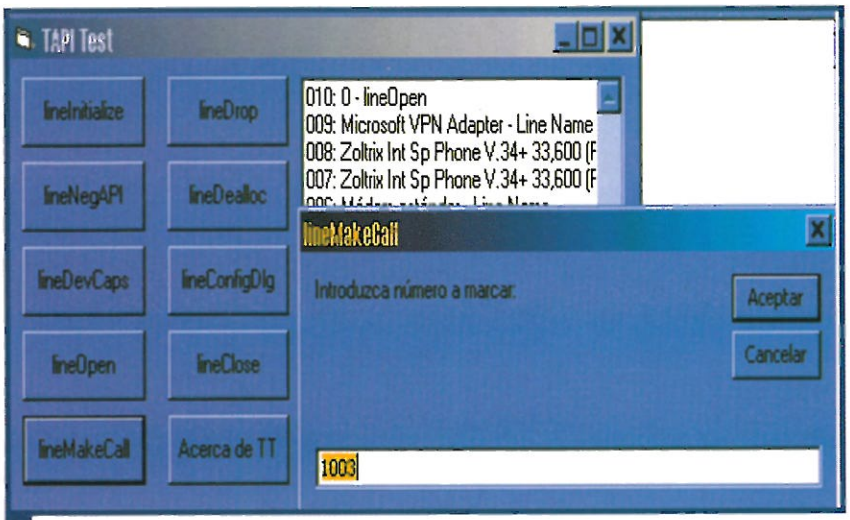


Figura 5.- La función *LineMakeCall* provoca la creación de una nueva llamada telefónica.

línea sin tener en cuenta el poseedor de la llamada, tiene que establecer los *flags* *LINECALLPRIVILEGE_OWNER* y *LINECALLPRIVILEGE_MONITOR*.

Una aplicación que haya abierto con éxito un dispositivo de línea siempre puede iniciar llamadas utilizando las funciones *LineMakeCall*, *LineUnpark*, *LinePickup*, *LineSetupConference* (con el parámetro *hCall* con el valor **NULL**), así como *LineForward* (asumiendo que puede hacerse dependiendo de las características del dispositivo, el estado de la línea, etc.).

PRIORIDADES

Pueden surgir conflictos si múltiples aplicaciones abren el mismo dispositivo de línea para el mismo *Media Mode*. Estos conflictos son resueltos a través de un esquema de prioridad mediante el que el usuario asigna las prioridades relativas a la aplicación. Esto se realiza por medio del panel de control de *Windows* o un menú de preferencias de una aplicación telefónica.

A continuación se enumeran algunos puntos que servirán de guía sobre el tema citado:

- Sólo la aplicación con prioridad más alta para un *Media Mode* dado recibe la propiedad de una llamada de dicho *Media Mode*.

- Aunque la propiedad normalmente se recibe cuando una llamada entrante llega primero, cualquier aplicación (incluso teniendo una prioridad más baja) puede adquirir la propiedad de la llamada usando las funciones *lineGetNewCalls* y *lineGetConfRelatedCalls*.

El usuario puede asignar las prioridades relativas a los módulos (aplicaciones y *DLLs*) que utilicen *TAPI*. La información de configuración resultante será guardada en el registro de *Windows*.

LINEOPEN

Una aplicación puede abrir un número indeterminado de líneas. Se puede utilizar la función *LineOpen* con el privilegio *LINECALLPRIVILEGE_MONITOR*, lo que significa que la aplicación sólo monitorizará, no poseerá, las llamadas entrantes de todas las líneas abiertas.

Se podría abrir una línea con el intento de poseer las llamadas entrantes especificando como privilegio *LINECALLPRIVILEGE-OWNER* y el *Media Mode* como **NONE**. La aplicación realmente podría especificar un número de *Media Mode* en este parámetro utilizando bits *OR* por cada *Media Mode* deseado. A continuación vemos un supuesto utilizando *bits OR* para *C++* y *Visual Basic*:

```
LINECALLPRIVILEGE_OWNER | LINECALLPRI-
    VILEGE_MONITOR
```

```
LINECALLPRIVILEGE_OWNER Or LINECALL-
    PRIVILEGE_MONITOR
```

Retomando el asunto, la aplicación recibiría las notificaciones de las llamadas entrantes para cualquiera de los *Media Modes* especificados y recibiría las llamadas de sus poseedores (dueños).

Existe el *flag* *LINEMEDIAMODE_INTERACTIVEVOICE* que permite que la aplicación reciba notificaciones de llamadas entrantes del tipo "voz interactiva" (llamadas de voz con una persona al final de la línea). Este es el uso más frecuente para la gestión de llamadas con *TAPI*.

DESARROLLO DEL PROYECTO PRJTAPI

El programa que vamos a realizar mostrará la utilización de las funciones más comunes que se utilizan con *TAPI* para la gestión de una línea telefónica.

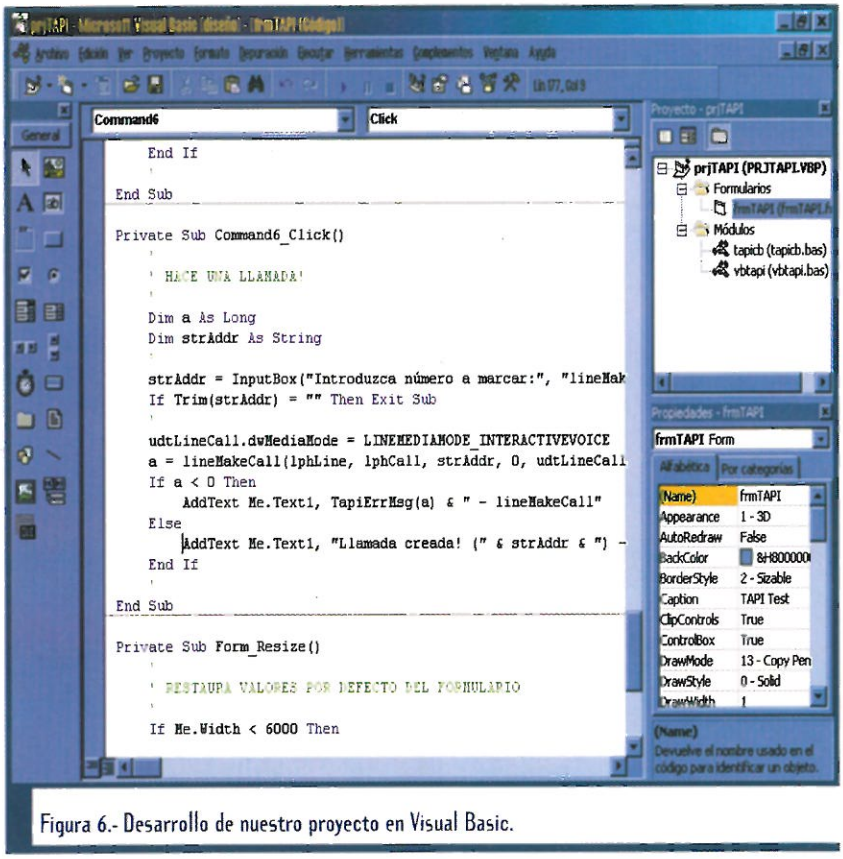


Figura 6.- Desarrollo de nuestro proyecto en Visual Basic.

El proyecto consta de un formulario y dos módulos bien diferenciados. El detalle de estos módulos es el siguiente:

Las líneas podemos monitorizarlas o poseerlas

- **tapicb:** Es el encargado de definir los mensajes de error de *TAPI* y varias funciones de ayuda para el soporte con *Visual Basic*.

- **Vbtapi:** En él se definen todas las constantes y se declaran todas las funciones de la *DLL TAPI32*. En *C++* tenemos la ventaja de que no es necesario realizar este trabajo puesto que accedemos a las funciones mediante la inclusión de los *headers* correspondientes.

El formulario **frmTAPI** albergará botones que realizarán diferentes acciones, desde la apertura de una línea hasta la creación de una llamada.

No vamos a hacer hincapié en el desarrollo de los módulos ya que para nosotros son meramente soporte de programación, aunque para los programadores más curiosos, a continuación se detallan sus aspectos más importantes.

MÓDULO TAPICB

Como ya se comentó anteriormente, en él se definen los mensajes de error de *TAPI*, además de contener funciones de ayuda extra.

Para definir constantes de error se utiliza la sintaxis *Public Const* seguida del tipo de error. Cada error suele llevar una numeración, tal y como se puede apreciar en el Listado 1.

Los privilegios determinan qué queremos hacer con las llamadas.

Entre las funciones de ayuda se encuentra *TapiErrMsg*, que se encarga de convertir números hexadecimales en mensajes de error comprensibles para el programador y a su vez, para el usuario. El código correspondiente se puede consultar en el Listado 2.

MÓDULO VBTAPI

En él se definen todas las constantes y se declaran todas las funciones de la *DLL TAPI32*. Para declarar funciones se utiliza la sintaxis *Declare Function*, tal y como se puede apreciar en el extracto del Listado 3.

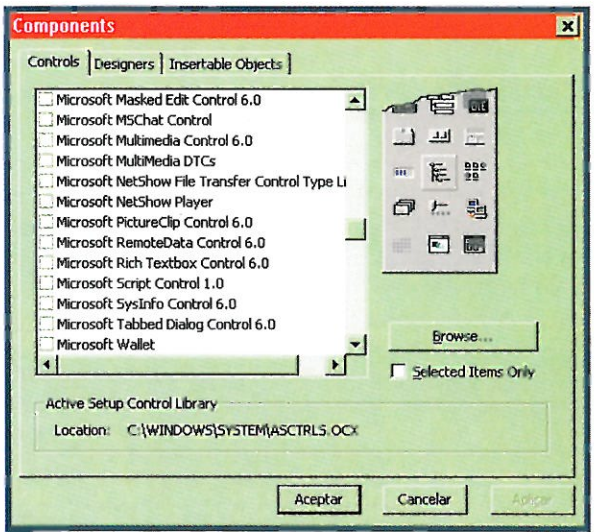


Figura 7.- Aunque busquemos un componente *TAPI* en *Visual Basic*, no lo encontraremos.

Listado 1: Constantes de error.

```
Errores de TAPI'
Public Const LINEERR_
     ALLOCATED =
    &H80000001
Public Const LINEERR_
    BADDEVICEID =
    &H80000002
Public Const LINEERR_
    BEARERMODEUNAVAIL =
    &H80000003
Public Const LINEERR_
    CALLUNAVAIL =
    &H80000005
Public Const LINEERR_
    COMPLETIONOVERRUN =
    &H80000006
Public Const LINEERR_
    CONFERENCEFULL =
    &H80000007
Public Const LINEERR_
    DIALBILLING =
    &H80000008
Public Const LINEERR_
    DIALDIALTONE =
    &H80000009
```

Listado 2: Función TapiErrMsg.

```
Public Function TapiErrMsg(dwMsg As Long)
  As String

'Convierte números hexadecimales
'en mensajes de error

Dim strTemp As String

strTemp = "Error TAPI desconocido!"

    Select Case dwMsg
Case LINEERR_ALLOCATED
' &H80000001

            strTemp = "LINEERR_ALLOCATED"

Case LINEERR_BADDEVICEID
' &H80000002

strTemp= "LINEERR_BADDEVICEID"

...
...

End Select

TapiErrMsg = strTemp & " ["
& Hex(dwMsg) & "]"

End Function
```

Cuando se declara una función es necesario incluir la referencia de la librería que la contiene. Puesto que en nuestro caso utilizamos *TAPI*, también debemos hacer referencia a la *DLL TAPI32*, tal y como ya apuntábamos en un apartado anterior.

Listado 3: Sintaxis Declare Function.

```
Declare Function tapiRequestMakeCall Lib "TAPI32.DLL" (ByVal
    lpszDestAddress As String,

    ByVal lpszAppName As String,
    ByVal lpszCalledParty As String,
    ByVal lpszComment As String) As Long

Declare Function tapiRequestDrop Lib
    "TAPI32.DLL" (ByVal hWnd As Integer,
    ByVal wRequestID As Integer) As Long
```

La función LineOpen se encarga de la apertura de líneas telefónicas TAPI

En la próxima entrega acabaremos el proyecto iniciado adentrándonos en su parte más importante: la programación del formulario principal del programa.

¿TE FALTA ALGÚN NÚMERO DE...?

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES

Nº 51: AGOTADO

BOLETÍN DE PEDIDOS

Rellene o fotocopie el cupón y envíelo a REVISTAS PROFESIONALES, S.L.
(Revista Sólo Programadores). C/ San Sotero, 5. 1ª Planta. 28037 Madrid
Tlf: 91 304 87 64. Fax: 91 327 13 03

Deseo que me manden los número/s.....
.....
.....

NOMBRE Y APELLIDOS:.....
.....

EDAD:..... PROFESIÓN: .....TFNO: .....
DOMICILIO: .....
CIUDAD: .....
PROVINCIA: .....C.P: .....

❑ **Precio por unidad: 975 pesetas** + 700 ptas. de gastos de envío.

FORMAS DE PAGO:

❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Talón bancario a nombre de REVISTAS PROFESIONALES. S.L
❑ Domiciliación bancaria
❑ Contra reembolso

DATOS DE DOMICILIACIÓN:

Banco.....
Domicilio.....
Nº de Cuenta.....|.....|.....|.....
Titular.....
Fecha.....

FIRMA

Programar un reproductor MM con Delphi (II)

Juan Luis Ceada Ramos
Programador en ARCABE Formación y Servicios Informáticos.

Continuamos en esta entrega el desarrollo de nuestro reproductor multimedia. Como prometimos en el número anterior, aprenderemos algunos trucos que pueden resultar de gran utilidad en futuros proyectos.

INTRODUCCIÓN

En esta ocasión vamos a aprovechar la facilidad que posee *Delphi,* para usar cualquier función de la *API* de *Windows,* para añadir nuevas funcionalidades a nuestra aplicación. Antes de comenzar, para hacerlo donde lo dejamos en el número anterior, recomiendo a los lectores que si no instalaron en su día los archivos con la ayuda del *API Win32*, éste es el momento más adecuado.

Figura 1.- Aspecto del nuevo reproductor.

IMPLEMENTACIÓN

Antes de empezar, recordemos que hay que tener instalado en el sistema el control *ActiveX Windows Media Player* (ver primer artículo de la serie para más información).

Con Delphi podemos acceder fácilmente a la API de Windows

En la Figura 2 se muestran únicamente el formulario principal y los paneles que hay distribuidos en dicho formulario. Cada panel tiene un código, cuyo significado podéis consultar en la Tabla 1. En ella, se indica el nombre de cada panel, el valor de su propiedad *Aling* y dentro de qué control se encuentra incluido.

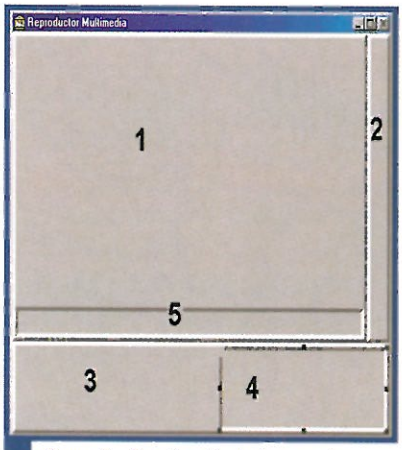


Figura 2.- Distribución de los paneles.

Todos los paneles tienen sus propiedades *BevelInner* y *BevelOuter* con el valor **bvNone**, aunque en la Figura 2 hemos cambiado estos valores para que resalten más los distintos paneles.

El valor asignado a la propiedad *Align* de cada panel puede hacer que nuestra aplicación se adapte muy bien (o que se convierta en un auténtico desbarajuste) ante un cambio de tamaño del formulario.

TABLA 1. Distribución de paneles.

Sección	Panel	Align	Contenido en
1	MediaP	*AlClient*	Formulario
2	BotonesP	*AlRigth*	Formulario
3	Pinferior	*AlBottom*	Formulario
4	PbotonesLista	*AlRigth*	Pinferior
5	Pbotones	*AlBottom*	*MediaP*

En la Figura 3 tenemos una captura del formulario principal de la aplicación en tiempo de diseño. A cada una de las zonas se le ha asignado un código. Iremos explicando cómo se ha implementado la aplicación poco a poco, empezando por la zona 1, y terminando en la 4.

TMMEDIAPLAYER

Lo insertaremos dentro del panel 1 (panel **MediaP**). Cambiaremos su nombre a **MP** y asignaremos a algunas de sus propiedades los valores que se muestran en la Tabla 2. El resto las dejaremos con los valores por defecto. Además, introduciremos código en los siguientes eventos:

- *OnEndOfStream:* se produce cuando termina la reproducción del fichero actual. En este momento, comprobaremos si en la lista de ficheros queda alguno por reproducir. Si es así, procedemos a su reproducción, en caso contrario no haremos nada.

Con TMediaPlayer, en diez minutos tendremos el 50% del proyecto realizado

- *OnNewStream*: se produce cuando se ha terminado de cargar un fichero, y va a dar comienzo su reproducción. En este momento, aprovechamos para actualizar las propiedades *Max* y *Frecuency* de la barra de desplazamiento. Lo haremos de tal forma que la barra siempre estará dividida en diez partes.

Puede consultar todas las propiedades y métodos disponibles en el archivo **TMediaPlayer.doc** incluido en el *CD-ROM* que acompaña a la revista. El código asociado a estos eventos se puede observar en el Listado 1.

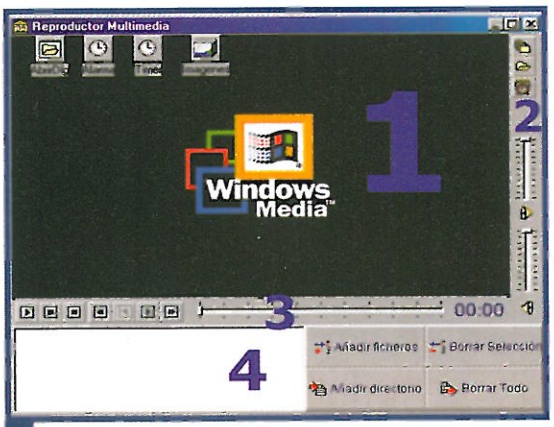


Figura 3.- Código asociado a cada zona.

BOTONES, VOLUMEN Y BALANCE

Aunque el control *TMediaPlayer* permite ajustar el volumen y el balance mediante las propiedades *Balance* y *Volume*, nosotros vamos a trabajar directamente con funciones de la *API* de *Windows*. Así aprenderemos cómo controlar estos parámetros mediante código, de forma que nos pueda servir para futuros proyectos.

El control del volumen lo haremos a "mano", usando la API de Windows

Para ello usaremos las funciones *WaveOutGetVolume (dispositivo, variable)* y *WaveOutSetVolume(dispositivo, valor)*. **Dispositivo** es un entero que identifica a la tarjeta de sonido. En sistemas normales (es decir, con una única tarjeta de sonido, es un **0**). Necesitaremos también usar variables auxiliares, que llamaremos **VolumenIzq** y **VolumenDer** (de tipo *Word*).

Lo primero que haremos (ver en el fuente el evento *FormCreate*) es obtener el volumen actual del sistema. Para ello basta con hacer lo siguiente:

```
waveOutGetVolume(0,@Volumen);
VolumenIzq:=Volumen;
//aquí se realiza un typecast
VolumenDer:=VolumenIzq;
```

donde **volumen** es una variable de tipo *dword*. El **0** indica que tomaremos el volumen del primer dispositivo reproductor de audio del sistema, es decir, de la tarjeta de sonido. Los 16 *bits* de mayor peso de **volumen** se corresponden con el volumen del canal derecho, y los *16* de menor peso, con el izquierdo. Nosotros nos quedaremos con el volumen del canal izquierdo, e inicialmente **VolumenIzq** y **VolumenDer** tendrán ese valor.

Los controles *TTrackBar* **VolIzqTb** y **VolDerTb** son los encargados de ajustar el volumen.

TABLA 2. Propiedades que se deben modificar.

Propiedad	Valor
lign	*AlClient*
AutoRewind	False
AutoSize	False
AutoStart	True
EnableContextMenu	False
Name	MP
ShowAudioControls	False
ShowCaptioning	False
ShowControls	False
ShowDisplay	False
ShowGotoBar	False
ShowStatusBar	False

Sus propiedades *Max* y *Min* valen **100** y **0**, respectivamente. Mediante una sencilla operación matemática, ajustaremos estos valores al rango **0-65535**, que son los valores extremos del volumen. En el Listado 2 podéis ver el código asociado a estos controles.

En el evento *OnChange* de ambos controles creamos un *DWORD* a partir de dos *WORD* (introduciendo desplazamientos cuando sea necesario). Este nuevo valor, se envía mediante *waveOutSetVolume* a la tarjeta de sonido, que ajusta su volumen al valor que le indicamos. Puesto que **VolumenIzq** y **VolumenDer** pueden tener diferentes valores, con esto conseguimos controlar el balance.

Por otro lado, los botones de arriba a abajo, realizan las siguientes funciones: mostrar/ocultar la lista de canciones, abrir un cuadro de diálogo para seleccionar el fichero *reproducir* y *activar/desactivar* el apagado automático (que ya vimos el mes pasado).

CONTROL DE LA REPRODUCCIÓN

Mediante los botones de la izquierda podemos iniciar la reproducción, detenerla, pausarla y movernos por la lista de reproducción de ficheros. El código asociado a estos botones es muy simple, un mero vistazo al fuente contenido en el *CD-ROM* debería bastar para entenderlo.

Antes de realizar una operación, hay que comprobar qué puede hacerse

La única precaución que se debe tener en cuenta consiste en comprobar que podemos realizar una operación antes de ejecutarla. Por ejemplo, este es el código asociado al botón **play:**

```
if MP.ReadyState=mpReadyStateCom-
    plete then
MP.Play;
```

Antes de llamar al método *play*, comprobamos que el control está listo para iniciar la reproducción (es decir, al menos hay un fichero cargado). Sin esta comprobación, al llamar a *play* sin haber ficheros cargados, se produciría un error.

Consulte el fichero TMMediaPlayer.pas del directorio Delphi\Imports.

Por cierto, al importar el control *ActiveX*, *Delphi* crea un fichero llamado *TMMediaPlayer_TLB.pas* en el directorio *Imports*, donde aparecen todas las constantes relacionadas con las diferentes propiedades del componente (como por ejemplo, la constante *mpReadyStateComplete*).

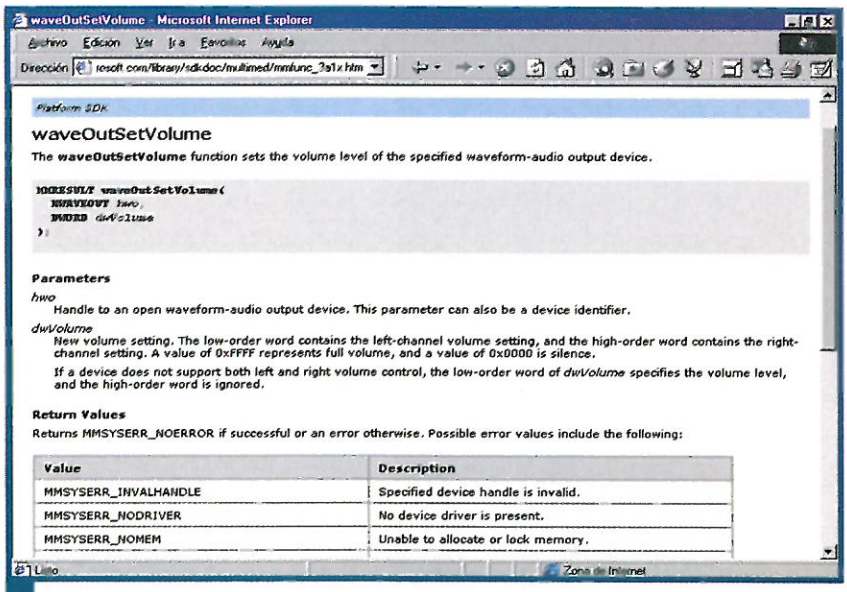


Figura 4.- Funciones de ajuste del volumen. *http://msdn.microsoft.com.*

Con respecto a la barra de desplazamiento, nos indica la posición actual de reproducción. Mediante un control *TTimer* (con la propiedad *Interval* igual a **1000**), que se activa cuando comienza la reproducción, y se desactiva al hacer *stop* (o cuando llegamos al final), vamos actualizando la posicion del control *TTrackBar*. Este es el código asociado al evento *OnTimer:*

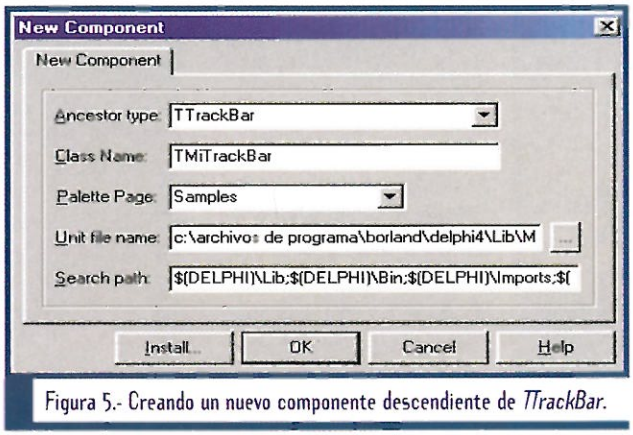


Figura 5.- Creando un nuevo componente descendiente de *TTrackBar*.

```
PosicionTb.Position:=Round(int(mp.
    CurrentPositio));
 //Actualizamos el label
 ActualizarLabel;
```

TMiTrackBar desciende de TTrackBar y añade los eventos OnMouseDown y OnMouseUp

La función *ActualizarLabel* toma la posición actual del fichero, la pasa a un formato de *minutos:segundos*, y actualiza el componente *TLabel* de la derecha. Si hacemos *clic* sobre dicho *TLabel*, podremos ver, alternativamente, el tiempo que nos queda, o el que llevamos actualmente de reproducción.

Pero además, el control *TTrackBar* permite variar a nuestro antojo la posición de reproducción. Pero este componente tiene un pequeño problema: no dispone de los eventos *OnMouseDown* y *OnMouseUp*, que precisamente son los que más nos interesan. Un rápido vistazo a los antecesores de este componente confirmará que desciende de *TWinControl*, que sí dispone de dichos eventos. Por lo tanto, el control *TTrackBar* dispone de ellos, sólo hay que hacerlos visibles.

Para ello creamos un nuevo componente, mediante la opción del menú **Component->Install Component**, cuyo antecesor sea un *TTrackBar*. Como nombre le podemos dar, por ejemplo, *TMiTrackBar*. Después editaremos el fuente de la unidad que *Delphi* crea, añadiendo en la sección *published* las siguientes líneas:

```
Property OnMouseDown;
Property OnMouseUp;
```

Y ahora sólo nos queda compilar e instalar el componente. Ya disponemos de un control *TTackBar* que se adapta a nuestras necesidades. En el Listado 3, podéis ver el código de la función *ActualizarLabel*, así como el asociado al *TTrackBar*.

LA LISTA DE CANCIONES

No se trata más que de un control *TListBox*. En dicho control se almacenan una serie de cadenas de texto. Cada una de ellas indica el nombre de un fichero, que posteriormente será reproducido por nuestra aplicación.

Para la lista de canciones usaremos un simple TlistBox

Esta lista está oculta inicialmente. Puede hacerse visible/invisible haciendo *clic* en el botón adecuado. Cuando la lista contiene valores, el usuario puede lanzar cualquiera de los ficheros que almacena haciendo un doble *clic*.

Mediante los botones situados a su derecha, podemos añadir ficheros a la lista (varios a la vez), o un directorio completo. También podemos seleccionar varios ficheros en el *TListBox* (este control permite realizar selecciones múltiples, mediante *control* & clic y *shift* & clic, propiedad *MultiSelect* igual a **True**), o borrarlos todos.

El código asociado a estos cuatro botones es muy simple. Sólo comentaremos algo acerca del botón **Añadir Directorio.** Este contiene una llamada a la función *SelectDirectory*, que necesita que se incluya en la unidad *FileCtrl* en la cláusula *uses*. Esta función lanza un cuadro de diálogo como el que se muestra en la Figura 6.

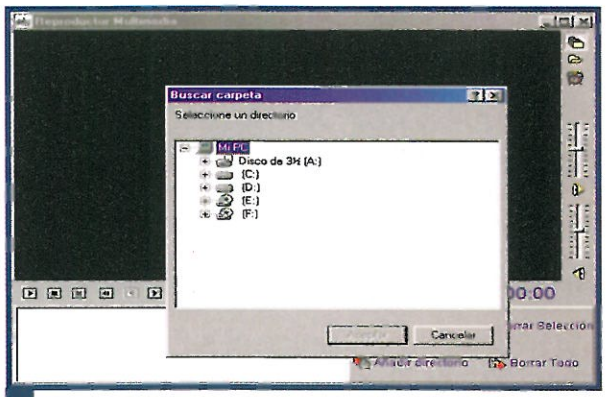


Figura 6.- Añadiendo un directorio completo a la lista de canciones.

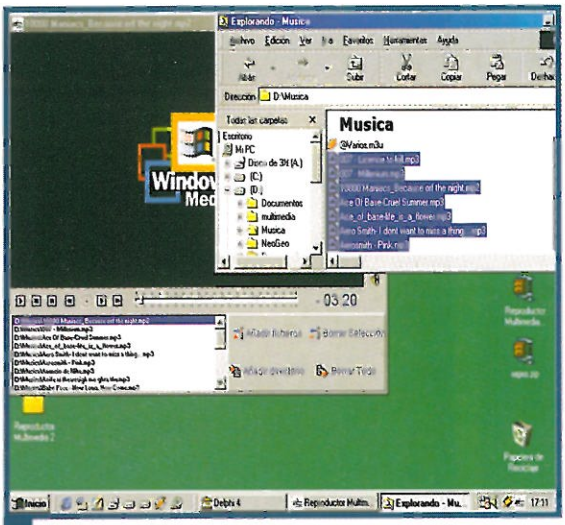


Figura 7.- Arrastrando ficheros sobre el reproductor.

dad *ShellApi* en el proyecto. Después hay que definir en la sección *private* un nuevo manejador de mensajes, que va a capturar todos los mensajes de tipo WM_-*DROPFILES*. Cuando se arrastran ficheros, *Windows* envía uno de estos mensajes a nuestra aplicación.

Este nuevo manejador de mensajes se encarga de averiguar cuántos ficheros se arrastran, y se van introduciendo los nombres de cada uno de ellos en la lista de canciones, para su posterior reproducción. Si no hay ningún fichero en reproducción se empieza por el primero de la lista.

HACER QUE ACEPTE FICHEROS

Hay algo que da un toque de profesionalidad a nuestras aplicaciones, y es relativamente fácil de implementar. Se trata de la posibilidad de que nuestro programa acepte los ficheros que sobre él se arrastren (por ejemplo, desde el explorador).

Definiremos un nuevo manejador de mensajes para aceptar los ficheros que nos sueltan

Todo el código necesario se puede observar en el Listado 4. En primer lugar, hay que incluir la uni-

Figura 8.- Aplicación minimizada mostrando el menú de reproducción.

LISTADO 1. Eventos de TMMediaPlayer

```
procedure
TFReproductor.MPNewStream
  (Sender:TObject);
begin
  PosicionTb.Max:=Round
  (Int(MP.duration));
  PosicionTb.Frequency:=
    PosicionTb.Max div 10;
end;

procedure
TFReproductor.MPEndOfStream
   (Sender:TObject; Result_:
    Integer);
begin
 if ListaLB.Items.Count>0
    then begin
   if ListaLb.ItemIndex<
     ListaLB.Items.Count -1
    then
   begin
     ListaLb.ItemIndex:=
       ListaLb.ItemIndex +
     1;
     ReproducirFichero(
        ListaLB.Items[
        ListaLB.ItemIndex]);
   end;
 end;
end;
```

LISTADO 2. Código asociado a los controles de volumen

```
//cambiar volumen canal
    izquierdo
procedure
   TFReproductor.
   VolumenIzqTbChange
(Sender: TObject);
begin
VolumenIzq:=Round( ((100 -
VolumenIzqTb.Position)*$FFFF)
 / 100);
waveOutSetVolume(0,
   (VolumenIzq
 shl 16) + VolumenDer);
end;

//cambiar volumen canal
    derecho
procedure
    TFReproductor.
    VolumenDerTbChange
(Sender: TObject);
begin
VolumenDer:=Round( ((100 -
    VolumenDerTb.
    Position)*$FFFF)
 / 100);
waveOutSetVolume(0,
(VolumenIzq shl 16) +
    VolumenDer);
end;
```

LISTADO 3. Código asociado a la zona 3

```
procedure TFReproductor.TimerTimer
  (Sender: TObject);
begin
  PosicionTb.Position:=
    Round(int(mp.CurrentPosition));
  ActualizarLabel;
end;

procedure TFReproductor.Tiempo
    LblClick(
  Sender: TObject);
begin
  //TiempoQueda: var global
  TiempoQueda:=not tiempoQueda;
  ActualizarLabel;
end;

//Actualiza las etiquetas
procedure TFReproductor.Actualizar
    Label;
var
 l:integer;
begin
if TiempoQueda then
 begin
 l:=Round(Int(MP.CurrentPosition));
 TiempoLbl.Caption := Format('+
  %2.2d:%2.2d ', [l div 60,
 l mod 60]);
end
else
begin
 l:=Round(Int(MP.Duration)) -
 Round(Int(MP.CurrentPosition));
 TiempoLbl.Caption := Format('-
 %2.2d:%2.2d ', [l div 60,
 l mod 60]);
end;
end;

procedure TFReproductor.Posicion
    TbMouseDown(....);
begin
  //aprovechamos el boton stop
  StopBtnClick(nil);
end;

procedure TFReproductor.Posicion
    TbMouseUp(...);
begin
  MP.CurrentPosition:=
  PosicionTb.Position;
  ActualizarLabel;
  //aprovechamos el boton play
  PlayBtnClick(nil);
end;
```

CONCLUSIÓN

Nuestro reproductor ya va tomando forma. Ahora sólo queda compilar, y probar su funcionamiento. Precisamente, durante las pruebas, me he dado cuenta de un pequeño inconveniente. Si la canción (o el vídeo!!) que estamos reproduciendo no es de nuestro agrado, lo normal es que rápidamente pasemos a la siguiente.

El próximo mes daremos algunas nociones de uso de la interfaz MCI

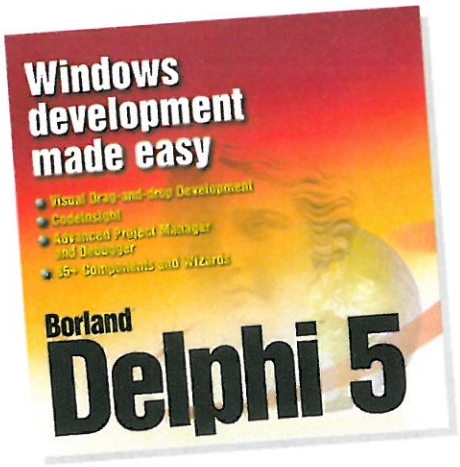
Sin embargo, si la aplicación está minimizada, esto se hace un poco pesado (hay que restaurarla, pulsar el botón siguiente, y volverla a miniminar). ¿No sería mejor poder pasar de canción mediante un simple *click* con el botón derecho sobre el icono minimizado de la barra de tareas?

Al hacer *clic*, debería aparecer un menú como el de la Figura 8, que permite reproducir, detener, pausar, etc... el fichero actual. En el próximo número veremos cómo hacer esto desde *Delphi*. Además, daremos algunas nociones de programación basada en la interfaz *MCI*, y puede que aún nos quede espacio para ver algún truquillo más...

LISTADO 4. Código necesario para que nuestra aplicación acepte los ficheros.

```
//Sección private del
    formulario
private
 procedure WMDROPFILES(var
    Message:
    TWMDROPFILES);message
    WM_DROPFILES;
.....

//Para aceptar que arras-
    tren ficheros sobre
    nuestra aplicación
procedure
    TFReproductor.
    WMDROPFILES(var
    Message:
    TWMDROPFILES);
var  NumFiles : longint;
     i : longint;
     buffer :
    array[0..255] of char;
begin
 //Vemos cuántos ficheros
    nos dejan}
  NumFiles :=
    DragQueryFile
    (Message.Drop,
    $FFFFFFFF, nil, 0);
 //Aceptar los ficheros
  for i := 0 to (NumFiles
    - 1) do
  begin
    DragQueryFile
    (Message.Drop,
     i, @buffer,sizeof
    (buffer));
    ListaLB.Items.Add(
     string(buffer));
  end;
  if not
     (MP.PlayState=mp
     Playing)
  then
   ReproducirFichero
       (ListaLB.Items[0]);
end;
```



Las novedades de Delphi 5

Juan Luis Ceada Ramos. Programador en ARCABE Formación y Servicios Informáticos

Evaluamos en este breve artículo las novedades más importantes de la nueva versión de *Delphi* que *Inprise* ha lanzado al mercado.

INTRODUCCIÓN

Inprise acaba de lanzar una nueva versión de su compilador *Delphi*. Aunque no representa un cambio tan grande como el que en su día se produjo al pasar de la versión 2 a la 3, sí se han incluido numerosas mejoras y novedades que harán la vida más fácil a los desarrolladores que lo hayan elegido como herramienta de trabajo.

En este artículo analizaremos las principales novedades que presenta *Delphi* 5 con respecto a *Delphi 4.03*, pero sin hacer referencia a ninguna versión en concreto. Sin embargo para ver qué elementos están disponibles en cada una de las diferentes versiones de *Delphi* (estándar, profesional y *enterprise*), se puede consultar la página http://www.inprise.com/delphi/productinfo/featurelist/.

	Estándar	Profesional	Enterprise
Procesador	166 MMX	166 MMX	166 MMX
Memoria	64 Mb	64 Mb	64 Mb
Disco (mínimo)	55 Mb	80 Mb	80 Mb
Disco (máximo)	120 Mb	185 Mb	230 Mb

EL ENTORNO

Una de las cosas que más llama la atención es el nuevo Inspector de Objetos, que incluye varias novedades.

Las propiedades "expandibles" presentan en un lado un signo +, que al ser pulsado despliega todas las propiedades asociadas. Conviene recordar que en la versión anterior se requería un doble *clic*. Además, en el propio Inspector de Objetos se muestran directamente los cursores, imágenes y colores dentro de las listas desplegables.

Otra característica importante se refiere a la posibilidad de organizar las propiedades y eventos en función de la categoría a la que pertenecen. En el Inspector de Objetos podemos ocultar/mostrar las categorías de propiedades/eventos que más nos interesen en un momento determinado y en función del tipo de proyecto abordado. De esta forma, podemos ocultar las propiedades que no suelen usarse, y mostrar solamente las que más nos interesan.

El Inspector de Objetos ahora es más funcional

Con respecto a los formularios, ahora se guardan en formato texto por defecto, aunque es posible seguir guardándolos en formato binario. Además, es posible hacer que, por defecto, los formularios que se añaden al proyecto NO se creen automáticamente. En *Delphi* 4, era necesario ir al menú **Projects->Options** para pasar los formularios añadidos de la lista *Auto-Create Forms* a la lista *Available Forms*.

DESKTOPS

Otra de las novedades se refiere a la posibilidad de organizar el *IDE* según las necesidades del usuario,

pudiendo almacenar varias configuraciones en un fichero con extensión *.DST*. La información que se almacena en estos ficheros incluye las ventanas que están visibles, su posición, así como qué ventanas están ancladas a otras. Además, se guardan las categorías de propiedades visibles y el estado de todas las barras de herramientas. De esta forma el desarrollador tendrá la posibilidad de elegir una u otra configuración en función de la tarea específica que está efectuando: edición, depuración, etc.

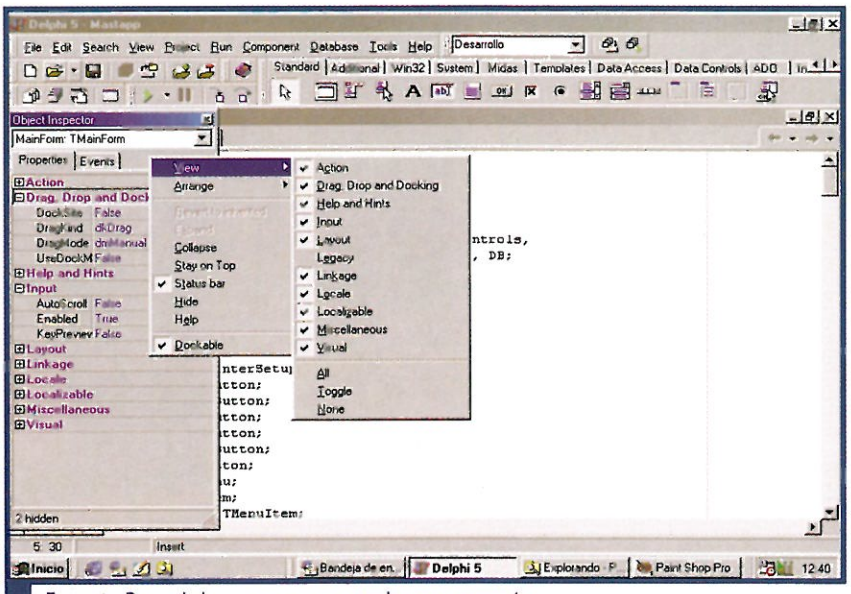


Figura 1.- Propiedades y eventos organizados por categorías.

Es posible utilizar un escritorio para desarrollar y otro para depurar

Para cargar un escritorio previamente almacenado lo más cómodo es recurrir a la lista desplegable situada en la nueva barra de herramientas *Desktops*. Gracias a esta nueva característica, podemos usar un escritorio para desarrollar la aplicación, y cambiar a otro completamente distinto a la hora de depurar, por ejemplo.

GESTOR DE PROYECTOS

El *Project Manager* también ha sufrido algunos cambios, destacando el hecho de poder copiar y pegar ficheros entre proyectos. Además, es posible arrastrar ficheros desde el explorador de *Windows* al proyecto. También es posible añadir directamente ficheros de recursos (*.RC*), que serán compilados a ficheros *.RES* sin necesidad de usar una herramienta externa.

MEJORAS EN EL DEPURADOR

Una de las novedades relativas al proceso de depuración consiste en poder visualizar una ventana con los registros de la *FPU/MMX*. Es posible arrastrar expresiones desde el código a la *watch list*, además de arrastrar una expresión a la ventana de la *CPU*, de tal forma que podemos conocer la dirección de memoria donde empieza dicha expresión.

La opción Run Until Return facilita la depuración

Una novedad muy útil es la opción *Run Until Retur*, que permite ejecutar una subrutina y vuelve a detener la ejecución cuando se retorna de la rutina llamada.

Es posible establecer más opciones para los puntos de ruptura. Por ejemplo, hacer que ante un punto de ruptura en particular no se detenga la ejecución del programa (sólo quedaría registrado en registro de eventos, menú **View -> Debug Windows-> Event Log**).

Se pueden agrupar los puntos de ruptura en grupos, de forma que al llegar a un determinado punto de ruptura, se active o desactive un grupo completo de *breakpoints*. Además, al colocar el ratón sobre uno de estos puntos, se muestran en un *tooltip* todas las opciones de dicho *breakpoint*.

OTRAS MEJORAS

También se han incluido dos nuevos asistentes. Éstos permitirán crear de un modo más sencillo aplicaciones para el Panel de Control, así como aplicaciones de consola.

VCL

Si bien el lenguaje (*Object Pascal*) no ha sufrido cambios en esta entrega, sí lo ha hecho la librería de componentes *VCL*. Se ha añadido un nuevo componente, *TApplicationEvents*, que permite simplificar la gestión de los eventos globales a la aplicación. Cada formulario puede tener un componente de este tipo. De esta forma, cuando se dispara un evento generado por la aplicación se ejecutará el código asociado al evento del componente *TApplicationEvents* correspondiente.

Se ha añadido una nueva página en la paleta de componentes: la página *Servers*. Contiene más de una docena de componentes que permi-

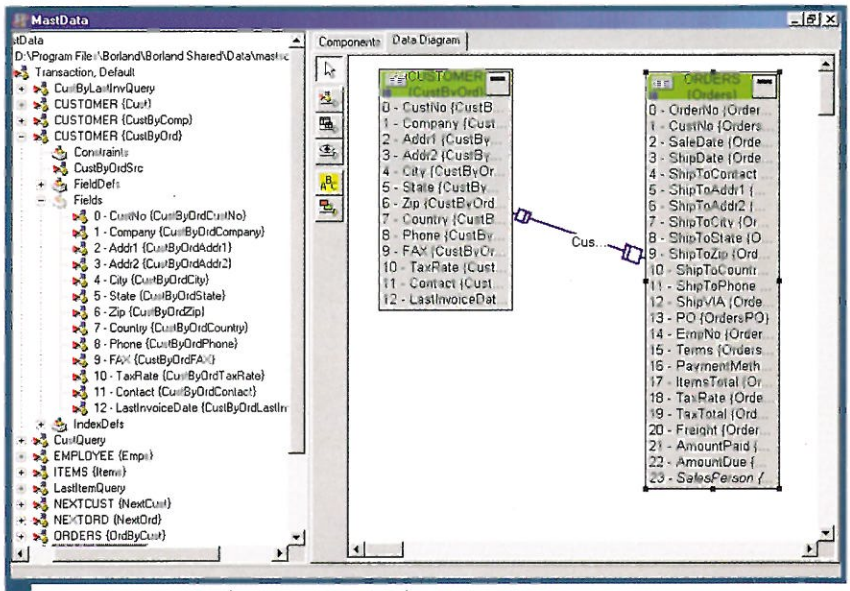


Figura 2.- Aspecto de los nuevos *DataModules*.

ten crear aplicaciones que se comuniquen con la suite *Office* de *Microsoft*. Hasta ahora había que recurrir a *OLE*, o a componentes externos para efectuar este tipo de tareas.

Se añaden componentes para conectar con las aplicaciones de la suite Office

Los componentes de *NetMasters* ahora aparecen en una página independiente (llamada *FastNet*). Estos componentes permiten desarrollar aplicaciones que hagan uso de los diferentes protocolos usados en *Internet* de una forma fácil y eficiente.

El nuevo componente *TWebBrowser* de la página *Internet*, reemplaza al control *THTML* de versiones anteriores. Se trata de un *wrapper* del control *ActiveX* que se incluye en la instalación del *Iexplore* 4 y 5. Permite incluir un verdadero navegador en nuestras aplicaciones.

Por supuesto, aparte de añadir nuevos componentes, se han mejorado algunos otros, como por ejemplo las listas de acciones, el componente *TToolBar*, *TListView*, *THeaderControl*, etc.

FRAMES

Una de las novedades más importantes de esta nueva versión de *Delphi* son los *frames*. Podemos pensar en un *frame* como una sección de un formulario que se puede reutilizar. Básicamente, permite crear componentes que a su vez estén formados por otros componentes (por ejemplo, un *frame* puede contener un *DBGrid* y un *TDBNavigator*). También es posible anidar *frames*.

Son muy útiles para organizar grupos de controles que se usan muy frecuentemente en las aplicaciones. Además, tienen otras ventajas: si tenemos un *bitmap* que se usa en varios formularios, podemos incluirlo en un *frame*, con lo que sólo se incluye en los recursos de la aplicación una única copia del *bitmap*, que puede ser usada en múltiples formularios.

Es posible transformar un *frame* en un componente. Cuando el usuario desee usar un *frame* sólo tiene que ir a la paleta de componentes, seleccionar uno e insertarlo en un formulario, como haría con cualquier otro componente. En resumen: un *frame* permite crear visualmente un grupo de componentes, definir un comportamiento por defecto, heredar de él, usarlo en múltiples aplicaciones y compartir recursos (como los *bitmaps*).

BASES DE DATOS

ADO

Se han añadido nuevos componentes que enlazan directamente con los conjuntos de datos *ADO* (*ActiveX Data Objects*) de *Microsoft*. Ahora tenemos los componentes *TADOTable*, *TADOQuery* y *TADOStoreProc* que acceden a fuentes de datos sin pasar por el *BDE*. Además, debido a la inclusión de esta nueva tecnología, se han añadido nuevos tipos de datos en la *VCL* (como *TVariantField*).

INTERBASE EXPRESS

Esta nueva colección de componentes permite acceder directamente a bases de datos *Interbase*, sin pasar por el *BDE*, lo que supone un importante aumento en el rendimiento de las aplicaciones.

NUEVOS DATAMODULES

Los antiguos *TDataModules* han sufrido un completo lavado de cara. Ahora se han convertido en una herramienta visual que hace más fácil crear, documentar y mantener las bases de datos. Se dividen en tres áreas.

La vista de componentes es la misma que en versiones anteriores. La vista de árbol (ver Figura 2) muestra la jerarquía existente entre los componentes del *TDataModule* (como por ejemplo, entre una tabla y sus campos, o entre una sesión y un *TDatabase*).

La vista de diagrama (ver Figura 2) muestra las relaciones entre los

componentes situados en el *DataModule*. Si por ejemplo, creamos en esta vista una relación maestro detalle (mediante simples operaciones de arrastrar y soltar), automáticamente se rellenarán las propiedades adecuadas (igual que si lo hubiésemos creado "a mano").

MIDAS

Permite crear aplicaciones distribuidas, de tal manera que la comunicación entre clientes y servidores puede realizarse a través de múltiples protocolos. Además de los componentes que permiten realizar las conexiones usando *COM*, *DCOM*, *CORBA*, etc., en esta versión se ha incluido el componente *TWebConnection*, que permite que las conexiones se realicen mediante el protocolo *HTTP*. Esto significa que podemos usar este componente para conectar a través de *firewalls*. Además podemos usar servidores seguros (que usan tecnología *SSL*).

Ahora las conexiones a servidores MIDAS pueden hacerse usando HTML

Los nuevos componentes *Internet Express* permiten desarrollar aplicaciones *Web* en las que los navegadores interactuen directamente con los datos provenientes de un servidor de aplicaciones *MIDAS* (con los componentes *TXMLBroker* y *TMIDASPageProducer*).

OTRAS MEJORAS

CORBA

Los componentes *CORBA* han sido actualizados para trabajar con el *ORB* de *VisiBroker for C++* versión 3.32. Adicionalmente, el tráfico de mensajes es menor dado que los clientes *CORBA* no necesitan estar constantemente enviando mensajes al servidor para mantener la conexión.

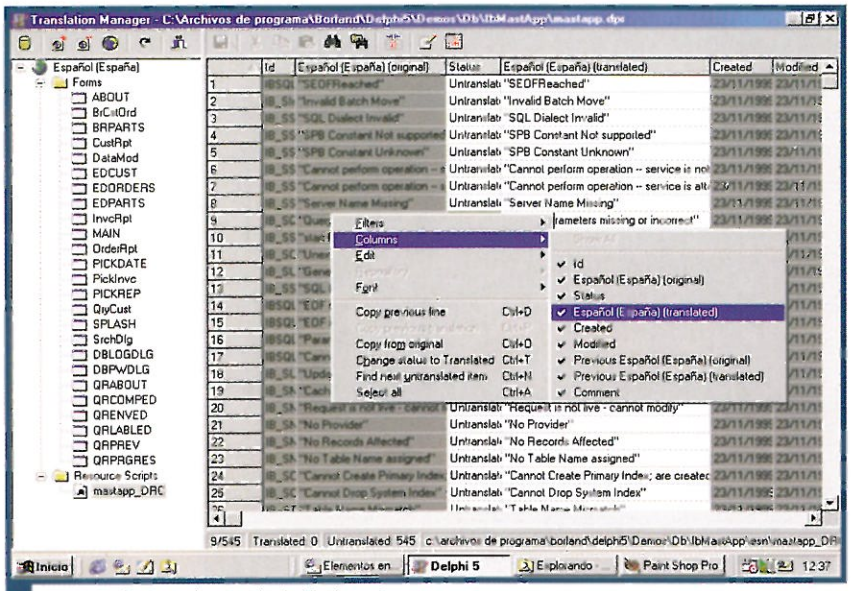


Figura 3.-Entorno Integrado de Traducción.

HERRAMIENTAS DE INTERNACIONALIZACIÓN

Se ofrece un nuevo conjunto de herramientas llamadas Entorno Integrado de Traducción (*Integrated Translation Environment - ITE*), que ayudan a desarrollar aplicaciones internacionales. Permiten traducir todos los recursos usados en nuestras aplicaciones.

TEAMSOURCE

TeamSource es una nueva herramienta para la gestión de proyectos compartidos, que ayuda a los equipos de desarrollo a controlar sus tareas diarias en un entorno de desarrollo compartido.

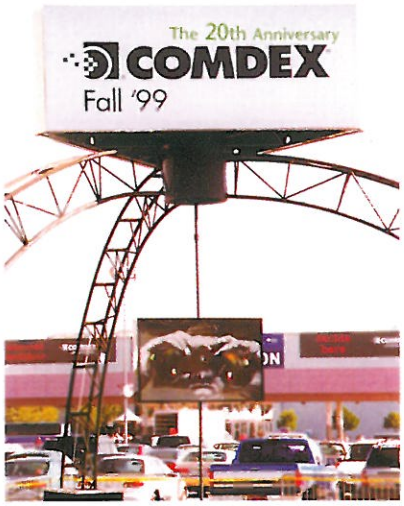
LISTAS DE TAREAS

Estas listas (*To-Do Lists*) permiten mantener una lista de tareas pendientes que desarrollar en el marco del proyecto activo. Se pueden añadir elementos a esta lista y asignarles prioridades, cambiar el estado en el que se encuentran, etc. También es posible añadir tareas a la lista desde el código fuente, mediante comentarios. El siguiente ejemplo añade una tarea pendiente, creada por el programador JLCeada, con prioridad 1:

```
{TODO 1 –oJLCeada : Arreglar
    fallo al pulsar F5}
```

CONCLUSIÓN

En este artículo sólo mostramos algunas de las múltiples novedades que incorpora *Delphi* 5. Sin duda alguna, *Delphi* es en estos momentos una de las mejoras herramientas de desarrollo del mercado. Sólo esperamos que esta vez *Inprise* se haya tomado las cosas con calma, y al contrario de lo que sucedió con *Delphi* 4, los parches para esta nueva versión tarden en aparecer. Para más información, consulte la página *web*: **http://www.inprise.com/delphi/productinfo/**.



COMDEX Las Vegas'99

El PC ha muerto, viva el PC

Joe M. Blacksmith

En su vigésimo aniversario la feria informática COMDEX, que se celebra en la ciudad norteamericana de Las Vegas en el mes de noviembre, ha marcado el punto de partida de lo que podríamos llamar la era *post-PC* o quizás mejor la era Internet.

INTRODUCCIÓN

Estamos asistiendo al final de una era de la informática y al nacimiento de una época en la que los ordenadores serán utilizados por el gran público. Cuatro han sido los protagonistas de esta edición de la más importante feria informática del mundo: *Bill Gates*, presidente de *Microsoft*, *Internet* y los nuevos *PC*'s, *Linus Torvalds*, creador de *Linux*, y *Scott McNally*, presidente de *Sun Microsystems*.

A partir de ahora cambiará sustancialmente lo que los compradores de tecnología informática buscarán en los nuevos ordenadores, tanto en prestaciones tecnológicas, como en servicios asociados e incluso en su diseño. Por esta razón es importante que una revista como Sólo Programadores analice estas nuevas tendencias y la polémica que las acompaña.

BILL GATES Y EL CONCEPTO DE WEB PERSONAL

En su conferencia *Bill Gates* comentó las últimas caídas de precios, la masiva conexión a *Internet* de los usuarios americanos de *PC*, el gran incremento del comercio *on-line* y las mejores posibilidades de distribuir audio y vídeo a través de *Internet*. Todo ello esta creando una atmósfera de grandes expectativas para todas las empresas grandes y pequeñas. Y en este sentido es vital saber cómo evolucionará la informática.

En opinión de *Gates* estamos pasando del concepto del ordenador personal al concepto del *Web* personal gracias al empleo del lenguaje de programación *XML*. El *Web* personal será una herramienta fantástica que permitirá que cada uno pueda adaptar *Internet* a sus necesidades específicas. Este hecho hará de la *Web* personal la herramienta de comunicación más poderosa inventada por el hombre.

Básicamente se trata de que los usuarios tengan acceso inmediato a la información que necesitan y el ordenador "aprenda" las necesidades del usuario para presentarle lo más importante. Como ejemplo de una *Web* personal se presentó un nuevo modelo de coche *Ford* dotado de sistema de navegación *GPS* y conexión a *Internet,* lo que permite descargar la ruta a seguir, calcular la gasolina a consumir, establecer la situación de las estaciones de servicio e incluso, si se necesita hacer una revisión del coche el ordenador, concertará la cita con el taller más adecuado. A través de este ejemplo es fácil deducir que en no mucho tiempo se generalizará el

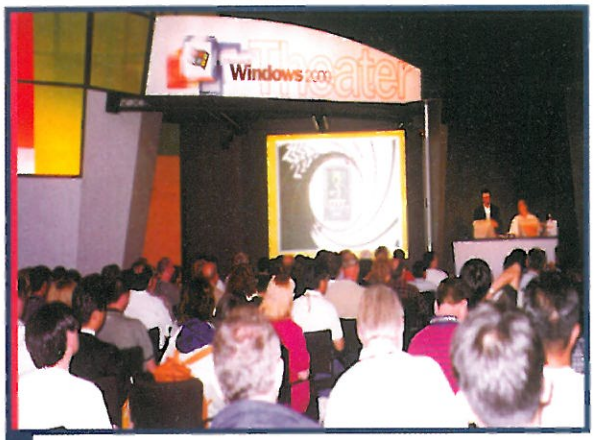


Figura 1.- *Microsoft* presentó la última versión de *Windows* 2000 previa a su lanzamiento comercial, que será en el primer trimestre del próximo año.

uso de la informática en los automóviles.

El concepto de *Web* personal es también aplicable al mundo empresarial en donde el establecimiento de procedimientos de búsqueda y almacenamiento de archivos y mensajes así como el uso de identificadores personales permitirá que los trabajadores del conocimiento tengan acceso a la información necesaria en el momento adecuado.

INTERNET Y LOS NUEVOS PC

Este año el *PC* tradicional ha sido el gran ausente de *COMDEX*. No es que no hubiera ordenadores personales en los *stands* del Centro de Convenciones de Las Vegas. Lo que pasaba es que el centro de atención se desplazó hacia los distintos dispositivos *Internet* preparados para trabajar junto al *PC* o incluso para sustituirlo.

La mayoría de los grandes fabricantes de *PC* han presentado sistemas, destinados tanto para la empresa como para los consumidores en general, cuya característica principal era la facilidad de uso que los hacía más semejantes a un electrodoméstico que a un *PC*.

Así pues los denominados electrodomésticos de la información o infodomésticos han alcanzado ya la mayoría de edad. Estos dispositivos son básicamente ordenadores simplificados y pensados para su conexión a *Internet*.

Se presentaron muchos modelos en la feria como comentamos más adelante, pero el hecho más significativo tuvo lugar durante la conferencia de *Bill Gates* en la que presentó el nuevo *Web Companion*, un nuevo concepto de ordenador de mano con una pantalla de 10 pulgadas que utiliza, lógicamente, el sistema operativo *Windows CE* y que se conecta a *Internet* a través del servicio *MSN* de *Microsoft*.

También está previsto que *Acer*, *Philips*, *Thomson* y otros fabricantes lancen sus versiones de este dispositivo. Sin embargo no tenemos noticias sobre qué pasará en España, aunque el renovado énfasis de *Microsoft* en su servicio *MSN* hace pensar que pronto oiremos hablar del *Web Companion*.

Sorprendentemente una de las compañías que ha presentado un cambio de estrategia más claro ha sido *Compaq*. Sus previsiones establecen que en el año 2005 el 90% de sus ventas estará constituido por ordenadores de mano (*PDA*), teléfonos móviles multifuncionales, buscapersonas, terminales de sobremesa pensados para la conexión a *Internet*, mientras que sólo el 10 % de estas ventas serán de ordenadores de sobremesa, portátiles y servidores.

Como muestra se presentó en la feria el terminal *iPaq*, que tiene sólo dos conectores *USB* en la parte frontal y lleva asociado un portal creado por *Compaq* para atender las necesidades del usuario de este tipo de dispositivos. La compañía tejana hizo también demostraciones del prototipo de un ordenador de mano con sistema operativo *CE* y pensado para la conexión a *Internet*.

Por su parte *Dell* ha anunciado para principios del 2000 un dispositivo similar llamado *webPC* muy reducido de tamaño y también con sistema operativo *Windows CE*. La estrategia de esta compañía está también cambiando rápidamente y ya tiene previsto el lanzamiento de una serie de dispositivos cuyas dos características fundamentales serán ser compatible con *Internet* y con el *PC*.

Los planes de *HP* son similares. Su nueva presidenta, *Carly Fiorina*,

Figura 2.- Como siempre, los desarrolladores dispusieron de su propia área de presentaciones, en donde podían ver en funcionamiento los últimos programas de *Microsoft*.

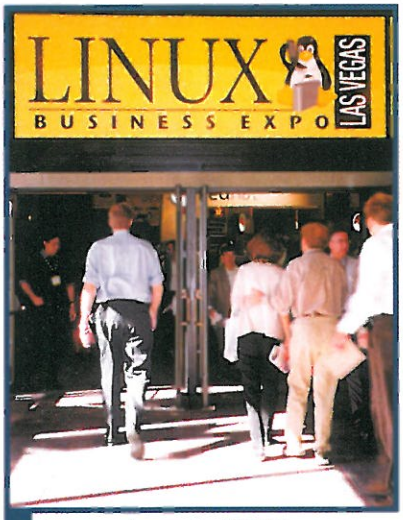


Figura 3.- Dentro de *COMDEX* se desarrolló la primera edición de *Linux Busines Expo* con un gran éxito de público y expositores.

presentó la nueva estrategia de la compañía en una concurrida conferencia en la empezó reconociendo el retraso de *HP* en descubrir que el motor de la industria informática está ahora en *Internet*. *IBM* se define como la compañía de *e-business* y *Sun* está vendiendo la mayoría de los servidores que hacen posible que *Internet* funcione, según reconoció *Fiorina*.

Tal vez por eso el futuro de *HP* incluye una gran campaña que transforme la imagen fría y distante que transmite el término "ciberespacio" en una imagen cálida, útil y personal de *Internet* lo cual facilitará su llegada a las familias, al gran público. *Fiorina* aprovechó su charla para anunciar un acuerdo con la compañía suiza *Watch* para desarrollar un ordenador con conexión a *Internet* que podrá llevarse puesto. Como conclusión señaló que las ventas se consiguen ahora con la conjunción de productos, infraestructura y servicios.

Uno de los grandes de la informática de consumo, que por cierto no desarrolla esta actividad en nuestro país, es *Sony*, en cuyos planes está el lanzamiento de una serie de dispositivos que facilitarán la conexión a la nueva *Internet* de banda ancha con lo que se pretende enviar música o incluso películas. Según *Nobuyuki Idei*, el nuevo presidente de *Sony*, se trata de poner en la palma de la mano de los usuarios toda la potencia del *PC*. El objetivo de la compañía es ser los grandes proveedores de entretenimiento a través de la banda ancha.

Por su parte tanto *Microsoft* como *Intel* hicieron mucho hincapié en que el *PC* está muy vivo y que no se trata de reinventar lo que ya está inventado sino de mejorarlo, haciendo los esfuerzos necesarios para simplificar su uso. *Microsoft* ha previsto el lanzamiento en el próximo año de la edición *Millenium* de *Windows 98*, mucho más fácil de usar, así como la tecnología Universal *Plug and Play* que permitirá la conexión generalizada a través de la red. *Intel* por su parte ha diseñado una nueva placa principal denominada *Flex ATX* con un tamaño mucho más reducido cuyo uso ha sido ya previsto por muchos fabricantes.

LINUX ES INEVITABLE

Entre los grandes conferenciantes de esta feria no podía faltar *Linus Torvals*, el ingeniero finlandés afincado ahora en Estados Unidos, que comenzó a trabajar en el *kernel* de *Linux* cuando era todavía estudiante universitario.

Su presentación corrió a cargo del presidente ejecutivo de *Linux Internetional, John "Maddog" Hall*, quien recordó la primera conferencia de *Torvalds* en Estados Unidos a la que acudieron apenas 40 personas; mencionó también una frase de esa primera conferencia: "*Linux* es inevitable", para terminar afirmando que el éxito de *Linux* está en que "es NUESTRO sistema operativo". Los aplausos fueron atronadores.

Al salir al escenario las dos grandes figuras de *Linux* se fundieron en un fuerte abrazo y *Torvalds* dijo: "Hace poco eramos unos chicos que jugábamos con pingüinos de felpa pero ahora hasta tenemos tarjetas de crédito con nuestro logo" a la vez que mostraba una tarjeta visa con el pingüino *Tux* como distintivo.

Linus Torvalds empezó su intervención agradeciendo a la comunidad de desarrolladores y usuarios de *Linux* su apoyo a la idea del código abierto. Hasta hace poco tiempo se trataba de un grupo de programadores que quería divertirse, últimamente el código abierto ha permitido que los usuarios hablasen con los programadores de manera que lo que se desarrollaba era lo que los usuarios necesitaban. Hasta gran-

Figura 4.- *Corel Corporation* aprovechó el foto de la feria *Linux* para presentar *CorelLinux*, el primer intento serio para llevar *Linux* a los usuarios no profesionales de la informática.

Figura 5.- En *COMDEX* nunca falta el espectáculo. En este caso se trata de cantar las excelencias de la fotografía digital.

des fabricantes como *HP*, *IBM* e *Intel* están apoyando el código abierto. "Pero la informática de sobremesa o personal sigue siendo la asignatura pendiente de nuestro sistema operativo", dijo *Torvalds* refiriéndose al hecho de que en general los usuarios tienen la idea de que *Linux* y *Unix* son difíciles de usar. " Y todavía queda algún tiempo para vencer esa dificultad si bien hay ya alguna luz en el horizonte" añadió.

Finalmente puntualizó que la gran ventaja de *Linux* es su escalabilidad que permite su uso tanto en los grandes servidores, su punto fuerte actual, como en ordenadores de mano tan pequeños como una caja de cerillas. La conferencia terminó con una ronda de preguntas y respuestas en la que se plantearon tres temas interesantes:

Pregunta: ¿Qué se está desarrollando en *Transmeta Corporation*, la empresa californiana donde presta sus servicios en la actualidad?

Respuesta: Estamos desarrollando una *CPU* inteligente en la que el *software* es muy importante. Y no puedo hacer ningún otro comentario hasta el 19 de enero del 2000 fecha del lanzamiento del producto.

Pregunta: ¿Qué pasaría si *Microsoft* decidiera lanzar su propia distribución de *Linux*?

Respuesta (después de alguna vacilación): Si *Microsoft* quiere hacer su propia distribución que la haga. Es posible, es legal y hasta les animaría a que lo hagan.

Pregunta: ¿Qué opina de los resultados de las pruebas efectuadas con los *"benchmarks"* de *Mindcraft*?

Respuesta: Este tipo de pruebas han mostrado aquellos aspectos del sistema operativo *Linux* que son mejorables. La comunidad *Linux* está siempre dispuesta a resolver los problemas y el nuevo *kernel* incluirá un sistema escalable de gestión de ficheros sensiblemente mejorado.

SOLUCIONES PARA TERMINAR CON MICROSOFT DE SCOTT MCNEALY

Son conocidas las críticas demoledoras que el presidente de *Sun Microsystems* hace de *Microsoft*. Pero esta vez aprovechando el auto del juez *Thomas P. Jackson* estableciendo, como hecho probado, que *Microsoft* actuaba como un monopolio, *Scott McNealy* propuso que el gobierno de Estados Unidos debía forzar la división de *Microsoft* en tres compañías que deben repartirse entre todos los empleados, del mismo modo que hacen los equipos deportivos cuando hay problemas, y a partir de ese momento las compañías deben competir entre sí.

En el camino hacia ese proceso *Microsoft* debe abrir las fuentes de sus *API's* (*Application Programming Interfaces*), adoptar una política de precios transparente y dejar de utilizar el dinero que obtiene con su actual situación de monopolio para intentar monopolizar otras áreas.

McNealy lamentó mucho que *Internet Explorer* domine la navegación por la Red y declaró solemnemente que *Microsoft* había destruido *Netscape*.

En su visión de futuro el presidente de *Sun* señaló que, si bien hasta ahora la industria informática ha estado centrada en el sistema operativo, esto no tiene porqué ser así en adelante. En el futuro la informática se parecerá a la telefonía "¿Conoce alguien el sistema operativo de su teléfono móvil? Ni falta que hace", concluyó.

Figura 6.- Los profesionales de todo el mundo acuden en masa a *COMDEX*. En esta ocasión se sobrepasó la cifra de 200.000 personas.

DNS a fondo (III)

Servicios con BIND 8.2.x

Vicente Antonio Sánchez Werner.
Desarrollador independiente.

En los artículos anteriores habíamos sentado las bases teóricas del funcionamiento del *DNS*, qué tipo de información podía contener y la forma de encontrarla. En éste implementaremos el servicio en nuestra máquina *Linux*.

INTRODUCCIÓN

El objetivo de este artículo es explicar cómo implantar un servidor *DNS* en nuestra máquina desempeñando diversas funciones útiles como pueden ser: caché de *DNS*, servidor *DNS* de una red local o cualquier otra aplicación que nos interese.

Existen diversos servidores de *DNS* pero en este caso hemos optado por *BIND 8.2.x* (en concreto 8.2.2 que es el último disponible en el momento de la redacción de este artículo) y lo hemos elegido por dos razones: es el más usado y el que más flexibilidad y prestaciones proporciona. El motivo de la elección de *BIND 8.2.2*, de entre otras versiones del mismo muy utilizadas (como *4.9.2*, *4.9.6* ó *8.2.1*), se debe a que estas versiones ya han sido abandonadas o están a punto de serlo, y es más sencilla de configurar que la 4.x.

INSTALACIÓN Y REQUISITOS

Para poder instalar *BIND 8.2.2* en nuestra máquina tenemos dos opciones, siendo la más sencilla para muchos usuarios la que utiliza el último paquete *BIND* compilado para una determinada distribución y su posterior instalación. De esta manera el usuario se ahorra el proceso de compilación y, en algunos casos, se simplifica la instalación del *software*.

La otra opción, más pesada pero más flexible, consiste en la instalación desde el código fuente, para lo que deberemos tener un compilador de *C* compatible *ANSI C*, una librería de *C* que cumpla con los estándares *ANSI C* y los paquetes *Bison* y *GNU-Flex* o equivalentes.

Si optamos por la primera opción, simplemente obtendremos el paquete actualizado y lo instalaremos en el ordenador con la utilidad de paquetes de nuestra distribución (*Yast*, *rpm*, etc..). Una vez hecho esto, procederemos a configurar el *software* en caso de que fuese necesario.

Para instalar el *software* desde los fuentes, el primer paso que se debe seguir consiste en conseguir la última versión disponible. En la dirección *http://www.isc.org/BIND.-html*, así como el paquete de documentación y un paquete con diversas utilidades ofrecidas por la comunidad de usuarios a lo largo del desarrollo de *BIND* y sus sucesivas versiones.

Una vez que tenemos en nuestra máquina el archivo con el fuente (*BIND-src.tar.gz*), lo descomprimimos y, después compilamos el código. Esta operación se realiza de la siguiente forma:

```
#tar -xzvf BIND-src
```

Con esta orden se descomprime el archivo y se crea un directorio llamado *src* en el que se encuentra la totalidad del código fuente de *BIND*, así como los archivos que son necesarios para su compilación en diversos sistemas operativos. En este punto entramos en el directorio y escribimos lo siguiente:

```
#make clean
#make all
#make install
```

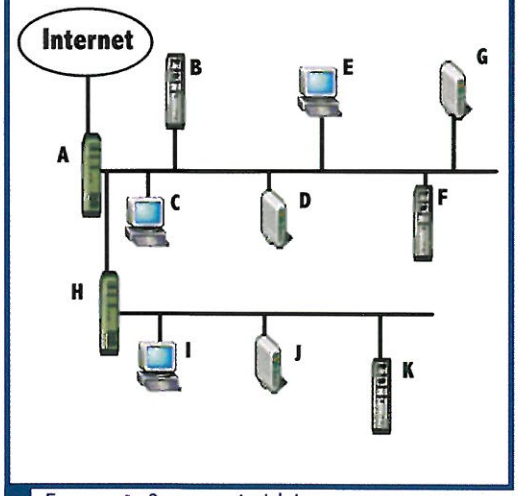
Con ello nos aseguramos de que no haya ningún resto de otra compilación anterior, luego indicamos que se compile (el sistema de compilación detecta automáticamente nuestro sistema operativo y usa el fichero *Makefile* adecuado) y finalmente le indicamos que instale los ejecutables, librerías y demás ficheros de *BIND*.

Este proceso no crea archivos de configuración ni modifica los archivos de nuestro sistema, por lo que estos pasos tendremos que realizarlos nosotros mismos.

Una vez que tenemos del *software* compilado, hay que observar la red actual y saber si ésta tiene las características necesarias para la función que necesitemos. En el caso de que deseemos administrar un dominio para nuestra red y que la red sea accesible desde *Internet*, hemos de averiguar varias cosas.

La primera es si la red está registrada o no, y la segunda decidir qué dominio queremos usar y averiguar si éste ya está registrado. En el primer caso, si las direcciones *IP* de nuestra red han sido asignadas por un proveedor de acceso a *Internet*, lo más normal es que estén registradas, por lo que tendremos que pedir al proveedor que nos delegue autoridad para poder administrar correctamente nuestras direcciones *IP* en el *DNS* (resolución inversa), ya que de lo contrario, la resolución inversa será llevada a cabo por el servicio *DNS* del proveedor, y dará un *FLQDN* erróneo o desfasado.

Si nuestra red va a conectarse directamente a *Internet* mediante un *carrier* (proveedor primario de acceso), éste puede proporcionarnos un grupo de direcciones, o podemos solicitarlas a la organización correspondiente, que en el caso de España y Europa es *RIPE* (*http://www.ripe.org*).



Esquema 1.- Organización del dominio *space.es*. Ver tabla 1.

La elección de dominio es algo más sencilla, ya que podemos acudir a *INTERNIC* y comprobar si el dominio elegido ya ha sido registrado. El problema aparece cuando efectivamente ya lo ha sido, o cuando ha sido registrado bajo un dominio de alto nivel diferente al nuestro.

El registro del dominio se realiza mediante el pago de una cuota anual a una de las empresas que se responsabilizan del mantenimiento y registro de los dominios en *Internet*. Si no es nuestro caso y sólo deseamos implantar el dominio de forma privada en una red, o si esta red no tiene conexión a *Internet* podemos obviar el registro de la red, el del dominio o ambos.

CONFIGURACIÓN DE BIND

Una vez que tenemos una red registrada y un dominio propio podemos proceder a configurar *BIND*. En un primer paso escribiremos los archivos de asignación de nombres y resolución inversa, y acabaremos escribiendo la configuración de *BIND* para nuestro dominio.

ORGANIZANDO EL DOMINIO

Una vez que tenemos un domino y una red registrados adecuadamente en *Internet*, lo que debemos hacer es observar cómo está organizada nuestra red o redes. A lo largo de este artículo se creará una red hipotética para el dominio *space.es* que abarcará las redes **205.234.24.0** y **205.234.209.0**, direcciones que nos han sido delegadas por nuestro proveedor de acceso.

La cesión de paquetes de direcciones *IP* en cantidades diferentes a las correspondientes a las clases A, B y C se realiza gracias al uso de un sistema de asignación llamado *CIDR*, que sustituyó a la asignación por clases.

Las redes están conectadas con un *router* y la primera de ellas es la que permanece conectada a *Internet*. Inicialmente, contamos con nueve ordenadores, seis de ellos en **205.234.24.0** y los restantes en la otra porción de la red. La organización de la red quedaría tal y como se muestra en el Esquema 1.

TABLA 1. Nombres y direcciones IP del esquema 1

Leyenda	Nombres	Direccion/es IP
A	stargate.space.es	205.234.24.1
B	nostromo.space.es	205.234.24.2
C	babylon5.space.es	205.234.24.3
D	galactica.space.es	205.234.24.4
E	sulaco.space.es	205.234.24.5
F	darkstar.space.es	205.234.24.6
G	enterprise.space.es	205.234.24.7
	warpspace.space.es	205.234.24.15
H	ws24.space.es	
	warpspace.space.es	205.234.209.1
	ws209.space.es	
I	rama.space.es	205.234.209.2
J	voyager.space.es	205.234.209.3
K	leonov.space.es	205.234.209.4

Lo primero que debemos hacer es decidir cómo vamos a denominar a las máquinas dentro de nuestro dominio, y luego asignarles sus direcciones *IP* donde podemos tener en cuenta el orden de llegada de las máquinas o su importancia y funcionalidad para asignarles una dirección u otra. Una vez hecho esto tendríamos la red organizada en direcciones *IP* y con nombres tal y como se ve en la Tabla 1.

CREANDO ZONAS

Llegados a este punto, lo mejor es comenzar a escribir los archivos que contendrán la información sobre el *DNS* (zonas). En este caso usaremos cinco archivos: **zn.space**, **zn.205.234.24**, **zn.205.234.209**, **zn.cache** y **zn.127.0.0**. El primero contendrá la correspondencia entre nombres y direcciones *IP*, los dos siguientes recogerán la correspondencia entre direcciones *IP* y nombres de la red, mientras que en *zn.cache* tendremos la lista de servidores *DNS* correspondientes al dominio raíz y *zn.127.0.0* cubrirá la red *loopback* (es la dirección *IP* que permite a una máquina conectarse consigo misma vía *TCP/IP* sin conocer su *IP* real).

La creación de los archivos de zona no requiere que se haga en un orden determinado, así como tampoco es preciso que los registros necesarios para describir una zona estén en un determinado orden. En cualquier caso usaremos un orden concreto para la creación de los archivos.

El primer fichero será **zn.space**, y en primer lugar crearemos el registro *SOA* para definir la autoridad para el dominio y los parámetros del mismo. El registro *SOA* para este dominio sería:

```
space.es.IN SOA babylon5.space.es
    dns.darkstar.space.es. (
          1    ;Numero de serie
      36000    ;Refrescar cada
               10 horas
       3600    ;Reintentar cada
               hora
     604800 ;Expirar tras una
               semana
      86400) ;Almacenar duran-
               te 1 día como
               mínimo
```

El primer parámetro que aparece es el nombre del dominio; éste aparece terminado por un '.' porque se refiere a su dependencia del dominio raíz '.'. El siguiente parámetro es **IN** e indica a *BIND* que el registro hace referencia a una red de tipo *Internet* (existen otras clases pero muy poco usadas). **SOA** indica que es un registro de tipo *SOA*, tras lo cual **babylon5.space.es**. indica el nombre del servidor *DNS* primario para este dominio.

El siguiente parámetro indica la dirección de *e-mail* de la persona responsable de este dominio; la sintaxis no es la habitual, sustituyéndose la habitual '@' por '.' por lo que en este caso el responsable del dominio sería

dns@darkstar.space.es.

Los paréntesis permiten distribuir la escritura de los parámetros del registro en varias líneas, mejorando su legibilidad y permitiéndonos incluir comentarios mediante el ';'. Estos parámetros permiten fijar la forma en que se propaga la información del dominio. El primer parámetro es un número de serie (forma que tienen los servidores esclavos de saber si ha cambiado la información del dominio), siendo un número de 32 *bits* sin signo.

El segundo parámetro indica a los servidores esclavos cada cuánto tiempo han de conectarse al servidor maestro para comprobar si la información está suficientemente actualizada. El tercer parámetro informa al servidor esclavo el tiempo máximo que puede retener la información del dominio si el servidor maestro no da servicio. En otras palabras, permite determinar durante cuánto tiempo es mejor dar información desfasada que no dar ninguna.

Finalmente el último parámetro indica a los servidores que reciben esta información el tiempo durante el cual pueden almacenarla en su caché. El tiempo en estos paráme-

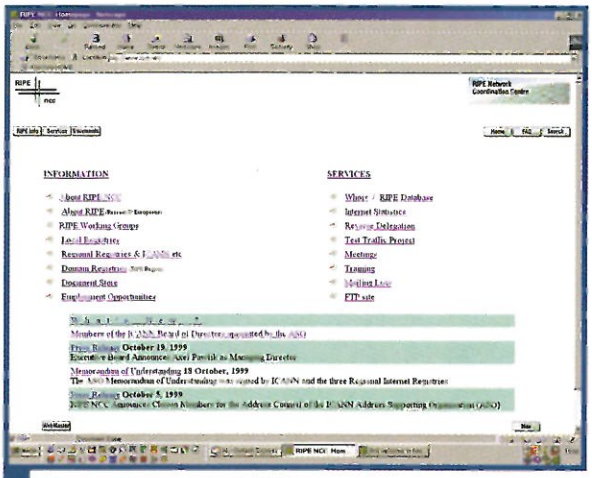


Figura 1.- Página *web* de RIPE.

tros siempre se expresa en segundos, por lo que no es extraño ver números muy grandes en estos registros.

Una vez definida la autoridad para la zona, debemos decir cuáles son los servidores de nombres para el dominio. Esto se hace con registros *NS*, que en nuestro caso serían:

```
space.es.INNSbabylon5.space.es.
space.es.INNSleonov.space.es.
space.es.INNSdns.soloprograma-
    dres.es
```

El primer parámetro que incluimos es el nombre del dominio para el que estamos estableciendo los servidores *DNS*, en este caso es **space.-es**. Los siguientes parámetros indican el tipo de red y el tipo de registro, que en este caso es **NS**. El último parámetro corresponde al nombre del servidor de nombres que estamos estableciendo.

Por cada registro *NS* sólo podemos asignar un servidor, por lo que siempre habrá tantos registros *NS* como servidores *DNS* delegados haya para nuestra red. En el registro *SOA* se indica cuál es el servidor maestro.

Finalmente, hemos de crear las asignaciones de nombres a direcciones *IP*, así como crear los alias para nuestras máquinas en caso de necesitarlas. Para ello usaremos los registros *A* y *CNAME*. El registro *A* tiene el formato siguiente:

```
<FLQDN><Clase>A<Direc-
ción IP>
```

LISTADO 1. Listado de registros A

```
;Maquinas de la red 205.234.24.0
localhost.space.es.INA127.0.0.1
stargate.space.es.INA205.234.24.1
nostromo.space.es.INA205.234.24.2
babylon5.space.es.INA205.234.24.3
galactica.space.es.INA205.234.24.4
sulaco.space.es.INA205.234.24.5
darkstar.space.es.INA205.234.24.6
enterprise.space.es.INA205.234.24.7

;Maquinas con varios interfaces de red
warpspace.space.es.INA205.234.24.15
warpspace.space.es.INA205.234.209.1
ws24.space.es.INA205.234.24.15
ws209.space.es.INA205.234.209.1

;Maquinas de la red 205.234.209.0
rama.space.es.INA205.234.209.2
voyager.space.es.INA205.234.209.3
leonov.space.es.INA205.234.209.4
```

De esta forma el listado de registros *A* para nuestra red quedaría como se observa en el Listado 1. Como podemos ver, tenemos todas las máquinas asignadas a una *IP*, excepto **warpspace.space.es.** Esta máquina tiene dos direcciones *IP*, además de su nombre canónico, porque está conectada a ambas redes por interfaces diferentes, gracias a lo cual nos permite comunicar ambas.

Con estos pasos ya tendríamos definidos los nombres de las máquinas y su asignación a las direcciones *IP* que tenemos en nuestra red. Sin embargo, es posible que una o más máquinas se conozcan por varios nombres y que (a diferencia de nuestro **warpspace.space.es**), sólo estén en una red. Estos nombres adicionales son alias y se añaden con el registro *CNAME*.

```
<alias><clase>CNAME<nombre
    canónico>
```

En nuestra red vamos a tener un servidor de *news*, un servidor de *FTP* central, un servidor *POP3* para el dominio así como una máquina que es la encargada del envío de correo al exterior. Podemos acceder a todas estas máquinas mediante sus nombres canónicos, pero preferimos acceder con un alias por sencillez y para prevenir cambios en el futuro derivados de movimientos de las máquinas. En este caso definiríamos los alias de la siguiente forma:

```
ftp.space.es.    IN
   CNAME   babylon5.space.es.
news.space.es.   IN
   CNAME   leonov.space.es.
pop3.space.es.   IN
   CNAME   darkstar.space.es.
mail.space.es.   IN
   CNAME   stargate.space.es.
```

En este punto ya tenemos definidos todos los registros necesarios para el funcionamiento correcto del *DNS*, pero tal y como lo hemos configurado tendríamos un problema con el correo electrónico que no llegaría a su destino porque no existe ruta alguna para él. Necesitamos una forma de decir cómo se va a mover el correo en nuestro dominio. Esto se consigue con el registro *MX* que indica qué máquinas están involucradas en la transmisión del correo al destinatario. La sintaxis del registro es:

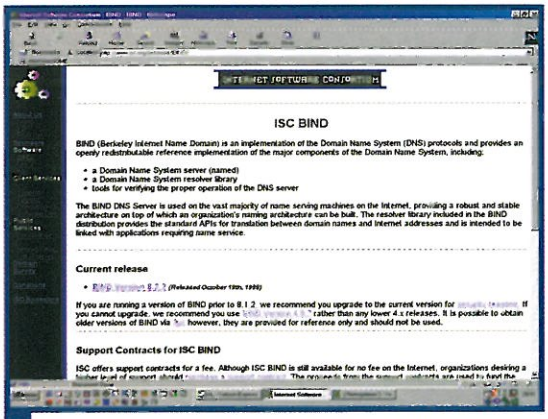


Figura 2.- Imagen de la página oficial de *BIND*.

```
<dominio><clase> MX
  <preferencia>  <FLQDN>
```

El dominio dice para qué dominio va a procesar o transmitir el correo esa máquina. La clase indica el tipo de red, *MX* es el tipo de registro y finalmente tenemos los dos parámetros más importantes: preferencia y *FLQDN*.

La preferencia es un número que indica la prioridad a la hora de enviar el correo a esa máquina: cuanto más pequeño es el número indicado mayor es la preferencia de la máquina especificada cuyo *FLQDN* hemos puesto. La preferencia en sí no tiene ningún significado en especial, sólo lo cobra en relación a las preferencias de otros servidores, y en caso de no haberlos, no tiene sentido alguno. En nuestra red el correo se va a mover de la siguiente forma:

```
space.es.INMX1000mail.
soloprogramadores.es.
space.es.INMX500stargate.space.es.
space.es.INMX0dark
     star.space.es.
```

Estos registros indicarían que la máquina a la que preferentemente deberían de llegarle los mensajes será **darkstar.space.es**. En caso de que no estuviera disponible, serán procesados por **stargate.space.es.** y en caso de que no estuviese disponible esta última lo recibiría **mail.soloprogramadores.es.**

Estas máquinas poseerán un agente de correo que según esté configurado retendrá el correo hasta que una máquina con menor número de preferencia esté disponible, o en caso de superar un límite de tiempo determinado devolverá el correo junto con un mensaje de error.

Con todos los registros creados, el fichero **zn.space** está ya completo. En este punto hemos de crear los ficheros de las zonas que permitirán la resolución inversa de los nombres. Para ello hemos de crear dos archivos similares a **zn.space**, pero que gestionarán la correspondencia entre direcciones y nombres del dominio para cada una de las redes. Estamos gestionando dos dominios: **24.234.209.IN-ADDR.ARPA** y el **209.234.205.-IN-ADDR.ARPA** que se corresponden con las sub-redes sobre las que tenemos autoridad.

Utilizamos un registro *SOA* para definir la autoridad de cada una de las zonas, en el que modificamos el nombre del dominio por el de la zona en cuestión:

```
24.234.209.in-addr.arpa.    IN
  SOA     babylon5.space.es.
          dns.darkstar.space.es. (
  1       ;Numero de serie
  36000   ;Refrescar cada 10 horas
  3600    ;Reintentar cada hora
  604800  ;Expirar tras una semana
  86400)  ;Almacenar durante un
          día como mínimo
```

Este registro creará la zona de autoridad para el fichero **zn.205.-234.24** y para tener este registro para la otra zona (**zn.205.-234.209**), sólo tendríamos que modificar el dominio inicial.

Una vez definida la zona debemos establecer qué servidores *DNS* han de dar el servicio:

```
24.234.209.in-addr.arpa.
   IN    NS    babylon5.space.es
24.234.209.in-addr.arpa.
   IN    NS    leonov.space.es
24.234.209.in-addr.arpa.
   IN    NS    dns.soloprogramado-
               res.es
```

Para la otra red sólo tendríamos que modificar el dominio inicial, ya que los servidores *DNS* son los mismos para ambas. En este punto asignaríamos direcciones *IP* a nombres de máquinas pero el registro *A* no nos sirve, por lo que hemos de usar el registro *PTR* para realizar la asignación y debemos usar la siguiente sintaxis:

```
<FLQDN en in-addr.arpa.> <Clase>
   PTR <nombre canónico>
```

Los registros *PTR* para nuestra primera zona tendrían el siguiente aspecto:

```
1.24.234.205.in-addr.arpa.
   IN    PTR    stargate.space.es.
2.24.234.205.in-addr.arpa.
   IN    PTR    nostromo.space.es.
3.24.234.205.in-addr.arpa.
   IN    PTR    babylon5.space.es.
4.24.234.205.in-addr.arpa.
   IN    PTR    galactica.space.es.
5.24.234.205.in-addr.arpa.
   IN    PTR    sulaco.space.es.
6.24.234.205.in-addr.arpa.
   IN    PTR    darkstar.space.es.
7.24.234.205.in-addr.arpa.
   IN    PTR
    enterprise.space.es.
15.24.234.205.in-addr.arpa.
   IN    PTR    warpspace.space.es.
```

Juntando el registro *SOA*, los registros *NS* y los *PTR* en el mismo archivo, tendremos el archivo **zn.205.234.24** que se encarga-

rá de realizar la resolución inversa para esta subred. Ahora quedaría realizar las mismas operaciones para la otra zona y obtener el archivo **zn.205.234.209.**

Aquí ya tendríamos todo lo necesario para que nuestro dominio funcionase adecuadamente, pero tendremos que crear dos ficheros más: **zn.127.0.0** y **zn.cache.** El primero para realizar la resolución inversa de nombres para la red *loopback* y evitar problemas en el caso de que alguna aplicación trate de efectuar una resolución inversa de esta red. Su aspecto en este dominio y red sería:

```
24.234.209.in-addr.arpa.IN
    SOA  babylon5.space.es
     dns.darkstar.space.es. (
            1    ;Numero de serie
        36000    ;Refrescar cada
                 10 horas
        3600     ;Reintentar cada
                 hora
        604800 ;Expirar tras una
                 semana
        86400) ;Almacenar duran-
                 te 1 día como
                 mínimo
```

Finalmente, quedaría el archivo *zn.cache,* que contiene la información de los servidores *DNS* de alto nivel que son el punto de origen de la información. Este archivo puede encontrarse con el nombre *named.root* en el directorio *domain* del **ftp.rs.internic.net,** por lo que podemos bajarlo y renombrarlo convenientemente para que se adapte a la estructura de nombres que hemos usado para definir las zonas.

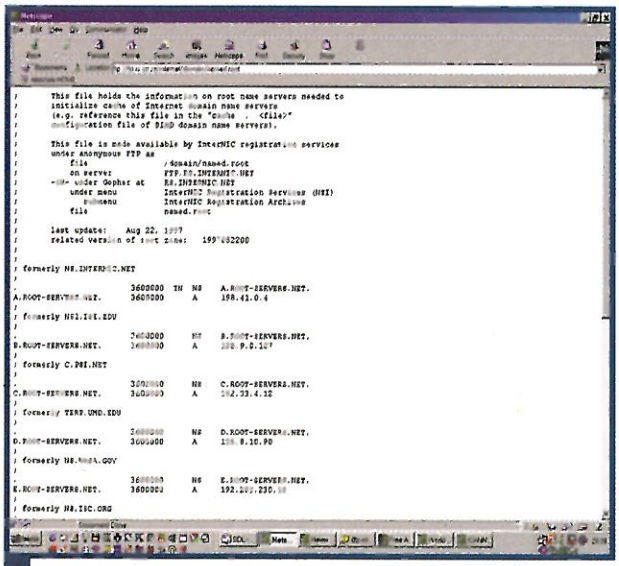


Figura 3.- Aspecto de named.root.

CONFIGURANDO BIND

Una vez creada la estructura del dominio, vamos a escribir el archivo de configuración de *BIND* para nuestras máquinas, además de modificar el arranque de las mismas para que *BIND* se inicie al arrancar. El archivo de configuración se llama *named.conf* y se suele colocar en el directorio */etc*. En primer lugar, configuraremos el servidor primario **babylon5.space.es,** luego el secundario que tenemos en la red (**leonov.space.es**) y asumimos que el tercer servidor (**dns.soloprogramadores.es**) es gestionado por nuestro proveedor de acceso.

La versión *8.x* de *BIND* que usamos permite crear los comentarios en tres formatos diferentes, siguiendo las normas del *C* (/* */, *C++* (//) y de la programación de *scripts* (#), pero no permite utilizar el ';' que se usaba en las versiones anteriores y que se usa en los archivos que contienen las zonas. El ';' tiene el significado final de comando, no de inicio de comentario.

Lo primero es decirle a *BIND* qué opciones deseamos usar y en qué directorio vamos a alojar los archivos de zona. Esto se consigue con la sentencia *options* de la que sólo puede haber una, que incluirá en su bloque de código las opciones que vamos a usar.

```
options {
directory "/usr/local/dns";
// aquí incluiriamos otras
// opciones
};
```

Este comando le indica a *BIND* que vamos a usar el directorio */usr/local/dns* para almacenar los archivos de las zonas. Además hemos de especificarle cuáles son los archivos que poseen la información de las zonas y decirle de qué tipo son. En este caso usamos diversas sentencias *zone*, que tienen la siguiente sintaxis:

```
zone "<dominio>" in{
      type tipo_de_archivo;
      file "<archivo que contiene
      la zona>";
};
```

En el caso de nuestro dominio la sintaxis de la zonas sería:

```
zone "space.es" in {
     type master;
     file "zn.space";
};
zone "24.234.205.in-addr.arpa" in
    {
     type master;
     file "zn.205.234.24";
};
zone "209.234.205.in-addr.arpa"
    in {
     type master;
     file "zn.205.234.109";
};
zone "0.0.127.in-addr-arpa" in {
     type master;
     file "zn.127.0.0";
};
zone "." in {
     type hint;
     file "zn.cache";
};
```

Estas opciones permiten configurar el servidor *DNS*, pero para *Linux* también hemos de modificar las opciones del *resolver* que trae incorporado y añadir los servidores *DNS* del dominio para que puedan utilizarse. El primer paso es añadir sus direcciones *IP* al archivo *resolv.conf*. Un ejemplo del mismo para este dominio sería:

```
domain space.es
nameserver 205.234.24.3
nameserver 205.234.209.4
```

Por lo que se refiere a los servidores se podría incluir como el primero la dirección **127.0.0.1**, ya que así se conectaría consigo misma para realizar la búsqueda de información. Esto se haría con el comando *nameserver 127.0.0.1*.

De esta forma se indicará a cada una de las máquinas *Linux* de la red (incluidos los servidores) que están en el dominio *space.es* y a qué servidores de nombres pueden interrogar.

El siguiente archivo que se debe modificar es **host.conf**, o si nuestra versión de *Linux* así lo requiere, en **nsswitch.conf**. En el primer caso deberíamos modificarlo para que tenga este aspecto:

```
order hosts, bind
multi on
```

Esto indica al ordenador que debe mirar primero en su archivo */etc/hosts* para encontrar la información y luego realizar la búsqueda de información vía el servidor *DNS*. Si se tratara de *nsswitch.conf*, debemos abrir este archivo y buscar la línea que tenga este aspecto o similar:

```
hosts:files nisplus nis dns
```

Debemos dejarla de la siguiente forma si sólo necesitamos el servicio de *DNS* y no *NIS* o *NIS+* (ambos son servicios de directorio que dan funcionalidades adicionales a la resolución de nombres):

```
hosts:files dns
```

Es posible que deseemos seguir accediendo a la funcionalidad de estos servicios, para lo que podremos dejar esta línea así:

```
hosts:files dns nisplus nis
```

Con estas modificaciones nos aseguramos de que nuestras máquinas accederán correctamente al servicio de *DNS*.

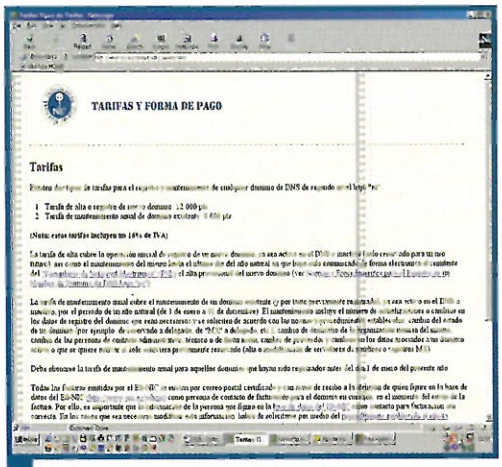


Figura 4.- Imagen de *www.nic.es*, el registro es para *.es*.

PUESTA EN MARCHA

Suponiendo que el archivo de configuración se encuentra en */etc/named.conf* se iniciaría el servidor *DNS* mediante:

```
#named
```

Si queremos estar seguros de que accede al fichero de configuración adecuado, siempre podemos especificarlo mediante la opción *-b* seguida de la ruta completa del archivo:

```
#named -b /etc/named.conf
```

Una vez que tenemos el servidor operativo, pasaremos a analizarlo con ayuda de *nslookup*, tal y como vimos en el artículo anterior. En caso de que no funcione, deberemos repasar su configuración, pero si ya es operativo podemos modificar el arranque de nuestra máquina para incluir la llamada a *named* como mejor nos convenga.

Con estos pasos tendríamos operativo nuestro servidor primario, pero no el secundario. Las diferencias entre el servidor primario y el secundario son pocas y se reducen a la posibilidad de modificar el contenido de las zonas y el origen de la información que sirven. Mientras el primario tiene acceso a los archivos originales, el secundario contacta con el primario para obtener una copia.

Para ello realizaríamos los mismos pasos que en el primario, pero modificaríamos el archivo *named.conf* para indicarle que es secundario. Esta modificación se realiza manualmente en cada sentencia *zone*, ya que es posible que un servidor secundario o esclavo sea primario de uno o más dominios y viceversa.

En nuestro ejemplo deberíamos modificar todas las líneas *type master*, a excepción de las dos últimas, que son las referentes a la zona *127.0.0.1* y al archivo de caché, que son exactamente iguales que las del servidor primario del dominio.

Finalmente, copiaríamos los ficheros de zona iniciales desde el primario (un *rcopy* o una transferencia por *ftp*) y pondríamos en servicio el servidor secundario de la misma forma que el anterior. En este momento tenemos nuestro dominio funcionando perfectamente, soportando el encaminamiento del correo electrónico y siendo perfectamente visible desde *Internet*.

SUSCRÍBETE A SÓLO PROGRAMADORES

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

- NOTICIAS
- MULTIMEDIA
- TEORÍA
- INTERNET
- DESARROLLO DE APLICACIONES

QUE NO TE LÍEN CON EL <CÓDIGO>

- COMUNICACIONES
- HERRAMIENTAS
- PROGRAMACIÓN
- ÚLTIMAS TENDENCIAS
- Y MUCHO MÁS...

BOLETÍN DE SUSCRIPCIÓN

Rellene o fotocopie el cupón y envíelo a REVISTAS PROFESIONALES, S.L. (Revista Sólo Programadores).
C/ San Sotero, 5. 1ª Planta. 28037 Madrid. Tlf: 91 304 87 64. Fax: 91 327 13 03

Quiero suscribirme a la revista SÓLO PROGRAMADORES desde el Nº.....
y beneficiarme de las condiciones de estas magníficas promociones:

❑ **SUSCRIPCIÓN ANUAL**
12 NÚMEROS + 12 CD-ROMs
AL PRECIO DE
9.360 ptas. / 56,25 €

❑ **SUSCRIPCIÓN ANUAL ESPECIAL ESTUDIANTES**
12 NÚMEROS + 12 CD-ROMs
por sólo 7.450 ptas. / 44,77 €

FORMAS DE PAGO:

❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Transferencia al Banco Popular Español. C/ Valdecanillas, 41.
Nº c/c: 0075/1040/43/ 0600047439
❑ Talón bancario a nombre de REVISTAS PROFESIONALES, S.L
❑ Domiciliación bancaria
❑ Contra reembolso

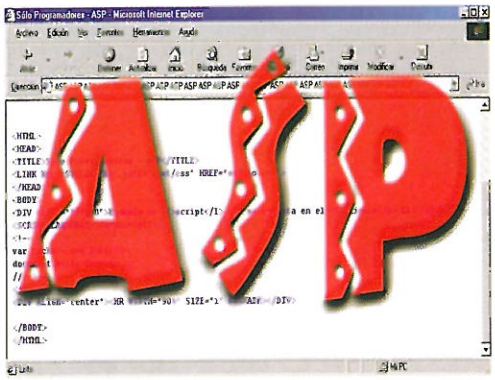
Soy antiguo suscriptor
❑ **Sí** ❑ **No**

PARA ENVÍOS AL EXTRANJERO SÓLO SE ADMITIRÁN LAS SIGUIENTES FORMAS DE PAGO:
❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Transferencia al Banco Popular Español. C/ Valdecanillas, 41.
Nº c/c: 0075/1040/43/ 0600047439
❑ Eurocheque conformado con un banco español a nombre de REVISTAS PROFESIONALES, S.L

NOMBRE Y APELLIDOS:.....
EDAD:..... PROFESIÓN: .....
TFNO: ..... DOMICILIO: .....
CIUDAD:..... C.P: .....PROVINCIA:.....

Promoción válida hasta agotar existencias

Datos de domiciliación:
Banco.....
Domicilio .....
Nº de Cuenta.....
Titular.....
Fecha .....
FIRMA



La tecnología ASP (III): Formularios y más formularios

Adolfo Aladro García. Analista programador

Hasta el momento hemos utilizado en nuestros ejemplos formularios, y más o menos se ha comenzado a ver la forma de manejarlos con páginas *ASP*. Ahora es el momento de profundizar en todos estos aspectos, de forma que seamos capaces de realizar aplicaciones *Web* avanzadas.

¿PASO DE DATOS POR FORMULARIO O POR URL?

Existen dos maneras de pasar información desde una página *Web* a otra: ya sea por el método *POST* o mediante el método *GET*. Normalmente el método *GET* está asociado al paso de parámetros a través de la dirección *URL* de la página:

```
http://127.0.0.1/ej.asp?p1=v1&p2=v2
```

Todo lo que se sitúa a la derecha del carácter de interrogación (?) constituye lo que se denomina la *query string*. Ésta se compone de una sucesión de pares, *parámetro = valor*, separados por *ampersands (&)*. Además tanto el nombre como el valor de esos parámetros está *url-codificado*, lo que significa que los caracteres especiales se han sustituido por **%N**, donde **N** es el valor en hexadecimal correspondiente al carácter especial.

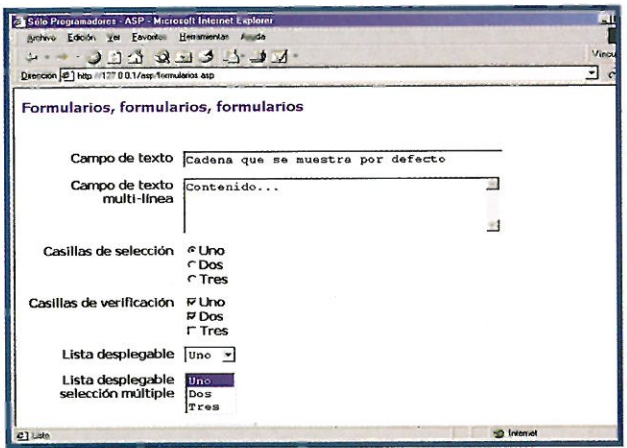


Figura 1.- Página *web* donde aparecen todos los elementos típicos de formulario.

El problema fundamental que tiene el paso de parámetros de esta manera reside en el hecho de que la *query string* se guarda en una variable de entorno, lo que implica que el tamaño máximo de los datos pasados vendrá fijado siempre por la capacidad de almacenamiento del entorno del servidor. Por ello, siempre que se desee mandar un número considerable de datos es preferente utilizar el método *POST* o lo que es lo mismo, los formularios.

ELEMENTOS DE UN FORMULARIO

Un formulario *HTML* puede tener campos de texto, listas desplegables en las que se puede

seleccionar sólo un elemento o varios, casillas de selección (sólo se puede seleccionar una), casillas de verificación (puede haber más de una seleccionada) y botones. La información que llega al servidor es diferente en cada caso y antes de procesarla es importante tener en cuenta las particularidades de cada tipo de elemento. Además de lo anterior, en aplicaciones complejas a menudo sucederá que nuestras propias páginas *ASP* no sólo tendrán que procesar los datos recibidos, sino que además tendrán que ser capaces de generar páginas con más formularios.

CAMPOS DE TEXTO

Dentro de esta categoría podemos encontrar tres tipos de elementos distintos: los campos de texto de una sola línea, los campos multilínea de edición y finalmente los campos de texto ocultos. En cualquiera de los tres casos anteriores el navegador siempre envía al servidor el contenido de esos campos en forma de cadena de texto, incluso cuando no han sido rellenados. En este caso, al consultar el contenido de ese campo se obtiene una cadena vacía.

```
<INPUT TYPE="text"
       NAME= "NOMBRE"
       VALUE="Madonna"
       SIZE="40"
       MAX LENGTH="60">
```

Los campos de texto de una sola línea se utilizan para enviar todo tipo de datos. Éstos son tratados como una cadena de texto y por lo tanto el programador tiene que encargarse de hacer las verificaciones manualmente en el caso de que sea necesario garantizar el tipo de los mismos.

Por ejemplo, si queremos asegurarnos de que el dato introducido es un número, tendremos que hacer las comprobaciones pertinentes a través de un *script* de cliente o bien de servidor. Como norma general este tipo de comprobaciones han de realizarse a través de un *script* de cliente siempre que se pueda y no se trate de un dato crítico susceptible de ocasionar algún tipo de desastre si es mal interpretado. De esta manera la carga del servidor será más ligera.

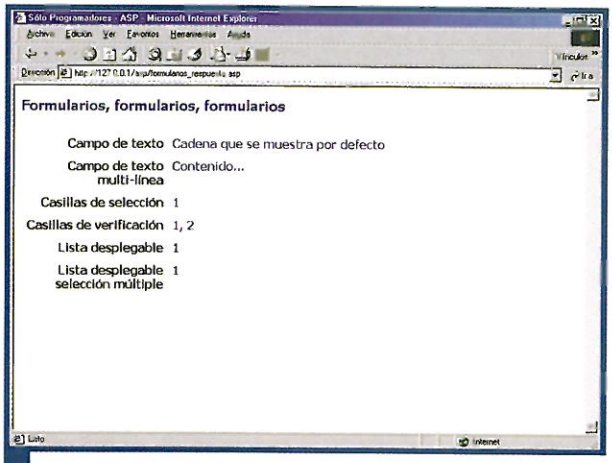


Figura 2.- Página que muestra los datos recibidos para cada uno de los elementos posibles de un formulario

- *SIZE*: Es el atributo que se utiliza para estable-cer el ancho del campo en número de caracteres. Pero esto sólo afecta a la visualización del campo.
- *MAXLENGTH*: Es el atributo que garantiza que, como máximo, el usuario podrá escribir un número determinado de caracteres.

```
<TEXTAREA COLS=38
          ROWS=4
          NAME="NOTAS"
          WRAP="Virtual">
Swim to the ocean floor
</TEXTAREA>
```

En general es preferible enviar los datos con el método POST

- *TEXTAREA*: Esta etiqueta se utiliza para presentar campos de texto en los que es posible escribir varias líneas.
- *COLS* y *ROWS*: Son atributos que especifican respectivamente el número de columnas y filas de caracteres.

Los formularios pueden generarse dinámicamente a partir de los datos mandados por otro de ellos

- *WRAP*: Es el atributo que sirve para determinar el tipo de salto de línea que se aplicará al campo.
- *OFF*: Este es el valor por defecto y hace que las líneas se muestren por la pantalla tal y como se escriben. Esto significa que si escribimos una línea más larga que el ancho del campo aparecerá una barra de desplazamiento para moverse de izquierda a derecha por el mismo.
- *VIRTUAL*: A medida que vamos escribiendo y la línea alcanza el ancho máximo del campo, se produce un salto. Este salto de línea es "virtual"

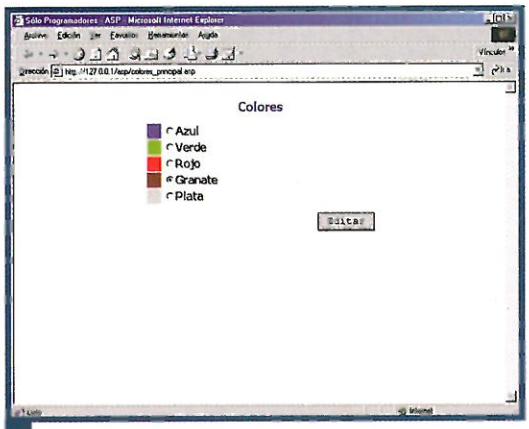


Figura 3.- Página principal de la aplicación que gestiona el archivo de colores.

en el sentido de que en realidad no se introduce el carácter salto de línea en el texto que es enviado al servidor.

- *PHYSICAL*: Este caso es igual al anterior con la salvedad de que el salto de línea sí que es introducido en el cuerpo del texto. Se utilizará en aquellos casos en los que queramos conservar exactamente la información del campo tal y como el usuario la introdujo.

Hoy en día una de las desgracias para todo desarrollador de aplicaciones *Web* es que cada navegador tiene sus particularidades y no todos interpretan los atributos de la misma forma. Con respecto a lo anterior cabe puntualizar que *Internet Explorer 3.0* aplica por defecto un salto de línea. Asimismo parece ser que el valor de esos atributos ha cambiado a partir de las versiones 4.x de *Netscape Communicator* e *Internet Explorer*. En estos casos se denominan: *off*, *soft*, *hard*.

Con frecuencia se da la circunstancia de que una página *ASP* tiene que mostrar los datos procedentes de un campo de texto multilínea. En estos casos hay que recordar que los saltos de línea introducidos sólo podrán hacerse visibles dentro de una página *Web* si son convertidos convenientemente a etiquetas *
*. De no ser así el navegador será incapaz de interpretarlos y no se mostrará correctamente la información que procede del campo de texto multilínea.

```
<INPUT TYPE="hidden"
       NAME="IDENTIFICADOR"
       VALUE="24">
```

Los campos de texto guardan cadenas de texto

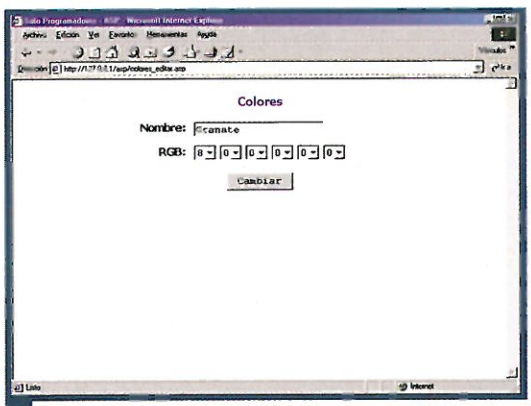


Figura 4.- Página que permite la edición de los colores.

Por último, los campos ocultos son uno de los recursos más utilizados cuando se quiere desarrollar aplicaciones *Web* complejas. Permiten transmitir información de una página a otra sin que el usuario sea consciente de ello. En este punto conviene recordar que debido a la naturaleza del protocolo *HTTP* las aplicaciones *Web* son aplicaciones sin estado. Cada vez que se realiza una petición al servidor es como si fuera la primera. Por eso cuando se desarrollan aplicaciones sofisticadas en las que cada página depende de las anteriores es necesario recurrir a un mecanismo para compartir la información. Además este mecanismo ha de ser transparente al usuario para que sea realmente efectivo. De ahí que los campos ocultos jueguen un papel tan importante.

LISTAS DESPLEGABLES

Las listas desplegables son otros de los elementos más utilizados en las páginas *Web*.

```
<SELECT NAME="LISTA"
        SIZE="3"
        MULTIPLE>
    <OPTION VALUE="1"
    SELECTED>Uno
    <OPTION VALUE="2">Dos
    <OPTION VALUE="3">Tres
</SELECT>
```

Existen dos tipos de listas desplegables: las que permiten al usuario elegir más de un elemento o aquellas otras en las que solamente se puede seleccionar una opción. La presencia o ausencia del atributo *MULTIPLE* determinará un caso u otro.

Desde el punto de vista de la programación de páginas *ASP* es importante hacer notar una diferencia considerable con respecto a los campos de texto. Cuando se selecciona un elemento de una lista desplegable y se envía el formulario correspondiente, la página *ASP* receptora recibirá el valor correspondiente al atributo *VALUE* de la opción seleccionada. Si se trata de una lista de selección

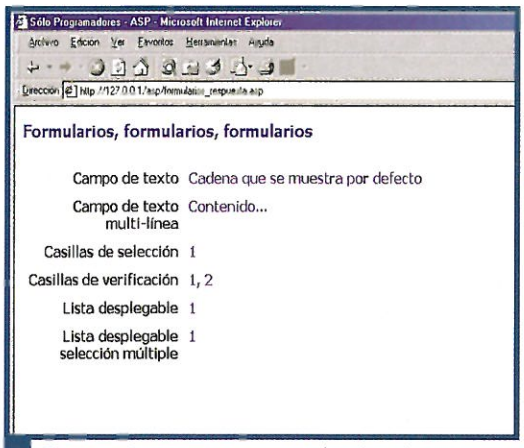


Figura 5.- Aspecto que presenta el archivo de texto *colores.txt*.

múltiple, se recibirán todos los valores escogidos. La propiedad *Count* del objeto devuelto por *Request.Form("LISTA")* permite conocer el número de opciones y así es posible recorrerlas todas de la siguiente forma:

```
for (i=0; i <
     Request.Form("LISTA").Count;
     i++) {

     Response.Write(Request.Form("
     LISTA")(i) + "<BR>");
}
```

Ahora bien, si no se escoge ninguna de las opciones, la página *ASP* receptora no recibirá nada procedente del formulario por parte de ese campo concreto. Esto no ocurre así con los campos de texto. Aunque éstos se dejen en blanco la página *ASP* receptora sigue recibiendo esos elementos.

Finalmente, en algunos casos determinados puede ser más conveniente tratar los valores procedentes de una lista de selección múltiple como un *array*. Cuando convertimos el resultado devuelto por *Request.Form("LISTA")* a una cadena de texto haciendo:

```
var lista = "" +
     Request.Form("LISTA");
```

obtenemos una lista de valores separados por comas. Este tipo de cadenas puede ser convertido rápidamente en *arrays* mediante la utilización del método *split*.

```
var lista_array =
lista.split(",");
```

El método *split* devuelve un *array* en el que cada elemento se obtiene de dividir la cadena original en tantos *tokens* o trozos como marque la cadena que el método recibe como parámetro, y que actúa de separador.

CASILLAS DE VERIFICACIÓN

Las casillas de verificación vienen a actuar a modo de listas desplegables de selección múltiple.

```
<INPUT TYPE="checkbox"
       NAME="CASILLA"
       VALUE="1"
       CHECKED>Uno<BR>
<INPUT TYPE="checkbox"
       NAME="CASILLA"
       VALUE="2"
       CHECKED>Dos<BR>
<INPUT TYPE="checkbox"
       NAME="CASILLA"
       VALUE="3">Tres
```

Todas las casillas de verificación que comparten el mismo atributo *NAME* se comportan como un todo. Cuando uno o varias de ellas están seleccionadas, se envían a la página *ASP* receptora de la misma manera que ocurre cuando se seleccionan una o varias opciones en una lista desplegable. Asimismo, cuando ninguna de ellas está activada, la página *ASP* receptora no recibe nada.

CASILLAS DE SELECCIÓN

Siguiendo con el paralelismo, estas casillas actúan como si fueran una lista de selección simple.

```
<INPUT TYPE="radio"
       NAME="CASILLA"
       VALUE="1"
       CHECKED>Uno<BR>
<INPUT TYPE="radio"
       NAME="CASILLA"
       VALUE="2">Dos<BR>
<INPUT TYPE="radio"
       NAME="CASILLA"
       VALUE="3">Tres
```

Las casillas de selección equivalen a una lista desplegable de selección simple

Al igual que ocurre con el caso anterior, todas las casillas que comparten un mismo nombre forman parte de un todo. Pero esta vez, el usuario solamente puede elegir una de ellas.

NUESTRO PRIMER EJEMPLO

Las páginas **formularios.asp** y **formularios_respuesta.asp** constituyen el primer ejemplo del artículo que nos ocupa. La primera de ellas contiene un formulario donde aparecen todos los diversos tipos de elementos que pueden ofrecerse en un formulario *HTML*, y la segunda se encarga de mostrar la información recibida. Observando los resultados expuestos se puede comprender perfectamente el funcionamiento de

cada uno de los elementos. Así por ejemplo, cuando en la lista desplegable de selección múltiple no se escoge ninguna de las opciones, aparece el valor **undefined** en la página de respuesta. Lo mismo ocurre con el conjunto de casillas de verificación.

APLICACIONES COMPLEJAS

Aunque las páginas anteriores son muy ilustrativas, la mayoría de las aplicaciones *Web* son bastante más complejas. No sólo los formularios son más sofisticados sino que además las páginas a las que llamamos tienen que generar más formularios de forma dinámica, a partir de los datos obtenidos de las páginas anteriores, consultas a bases de datos, lecturas de archivos, etc.

En este sentido las páginas **colores_principal.asp**, **colores_editar.asp** y **colores_editar_respuesta.asp** constituyen un pequeño ejemplo de cómo se construyen aplicaciones más complicadas. Al mismo tiempo introduciremos la utilización de componentes *ActiveX* del servidor, un tema que trataremos con profundidad en capítulos sucesivos.

Nuestra modesta aplicación representa un gestor de información. En un archivo de texto denominado **colores.txt** se almacenan datos acerca de varios co-lores.

```
Azul|0000FF
Verde|00FF00
Rojo|FF0000
Granate|800000
Plata|C0C0C0
```

Cada línea del archivo de texto contiene el nombre de un color seguido del carácter separador |. A continuación, en esa misma línea, se encuentra el código *RGB* del color. La página principal de la aplicación, **colores_principal.asp**, lee los datos del archivo de texto y muestra por pantalla una tabla donde aparecen cada uno de los colores al lado de una casilla de selección.

El Listado 1, que encontraréis en el *CD-ROM*, contiene el código fuente del *script* de la página principal de la aplicación que lee el archivo de texto y genera el código *HTML* correspondiente.

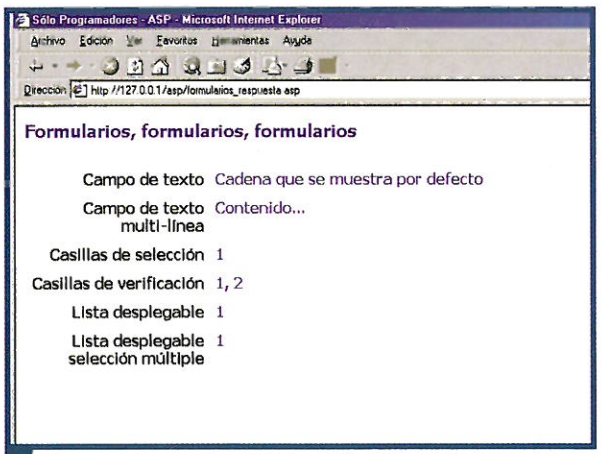


Figura 6.- Observando el código fuente en el navegador podemos ver el campo oculto que almacena el *RGB* del color a modificar.

Para leer el achivo de texto se utiliza el componente *ActiveX* de servidor denominado *FileSystemObject*. Los componentes *ActiveX* son objetos que están a disposición del programador y que sirven para tareas tales como leer, crear y/o modificar archivos de texto, acceder a bases de datos, conocer información concreta acerca de la petición *HTTP*, etc. La sentencia:

```
filesystemobject =
    new
      ActiveXObject("Scripting.File
      SystemObject");
```

crea una instancia del objeto *FileSystemObject* y a partir de ella podemos acceder a todas las propiedades y métodos de ese componente, mediante los cuales leeremos el archivo de texto.

El método *OpenTextFile* abre un fichero de texto y devuelve un objeto del tipo *TextStream*. El primero de los parámetros representa la ruta al archivo de texto y el segundo indica qué tipo de operación se va a realizar. En nuestro caso concreto ese valor será **1**, lo que significa que vamos a leer el archivo de texto.

Una vez que el archivo se ha abierto y tenemos un objeto del tipo *TextStream*, podemos leer todas las líneas del archivo de texto haciendo uso del método *ReadLine*. La propiedad *AtEndOfStream* tomará el valor **true** cuando se haya leído todo el archivo de texto.

Cada línea leída ha de ser procesada adecuadamente. Por un lado necesitamos extraer el código *RGB* del color correspondiente para hacerlo aparecer en la pantalla. Asimismo también es preciso extraer el mnemónico o nombre del color.

```
pos = linea.indexOf("|");
color_mnemonico = linea.subs-
    tring(0, pos);
color_rgb =
    linea.substring(pos+1,
    linea.length);
```

La variable **pos** almacena para cada una de las líneas la posición donde se encuentra el carácter

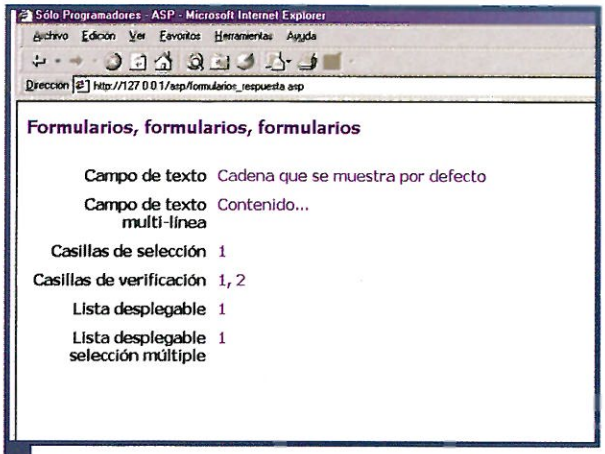


Figura 7.- Resultado de modificar el color "Granate" para crear el color "Morado personalizado".

separador y el método *substring* permite extraer la subcadena que se extiende desde una posición inicial hasta otra final dentro de la cadena que forma cada línea.

Cada color dará lugar a una casilla de selección en el que su valor será exactamente el mismo que constituía la línea de texto procedente del archivo.

El usuario podrá seleccionar cualquiera de los colores y al pulsar el botón **Enviar** se pasará a la página de edición de ese color: **colores_editar.asp**. En esta página se muestra un campo de texto con el nombre del color que editar y una serie de listas desplegables de selección simple para cada uno de los valores hexadecimales del *RGB*. El Listado 2, también incluido en el *CD-ROM*, muestra el código del *script* encargado de procesar los datos recibidos de la página principal.

Siguiendo un mecanismo similar al utilizado en el caso anterior se divide la cadena recibida para separar el nombre del color de su código *RGB*. Al mismo tiempo se extraen cada uno de los valores hexadecimales correspondientes al *RGB* del color. Para ello se utiliza el método *charAt*, el cual devuelve el carácter de una cadena de texto que se encuentra en una posición determinada.

Las listas desplegables que muestran todos los valores hexadecimales que pueden tomar los dígitos *RGB* han de generarse dinámicamente ya que inicialmente tienen que estar seleccionados los dígitos correspondientes al color enviado. El Listado 3, en el *CD*, ilustra el *script* que realizará esta tarea.

Finalmente, en esta página también se genera un campo oculto donde se almacena el código *RGB* del color que se va a modificar.

```
<INPUT TYPE="hidden"
    NAME="COLOR_RGB_ANTIGUO"
                VALUE="<%=
    col_rgb %>">
```

Este campo nos permitirá saber cuál es el color del archivo de texto original que se ha modificado.

La página **colores_editar_respuesta.asp** se encarga de recoger el cambio de color en el archivo de texto **colores.txt**. Para ello vuelve a hacerse uso del componente *FileSystemObject*. Se abre el fichero original y se lee línea a línea. Estas líneas se escriben al mismo tiempo en un archivo de texto temporal. Cuando se detecta el color original que se desea modificar, la línea que se escribe es la que se deduce de las modificaciones introducidas. Al final el archivo de texto temporal se copia al archivo original y la página se redirige automáticamente a la página principal de la aplicación. En ésta se verán los cambios introducidos.

Esta modesta aplicación muestra un repertorio de acciones que se aproxima bastante al que realizan las aplicaciones *Web* profesionales en muchos casos: formularios que generan más formularios y que han de procesar la información teniendo en cuenta fuentes de datos externas. Este modelo puede complicarse todo lo que sea preciso y a medida que esto ocurre tenemos que tener presente aspectos de seguridad que en nuestro ejemplo no hemos analizado. ¿Qué ocurriría por ejemplo si al modificar el nombre de un color introducimos en carácter |? Este tipo de casos han de ser detectados para evitar errores, bien a través de *scripts* de cliente o bien dentro de las propias páginas *ASP* encargadas de procesar los datos.

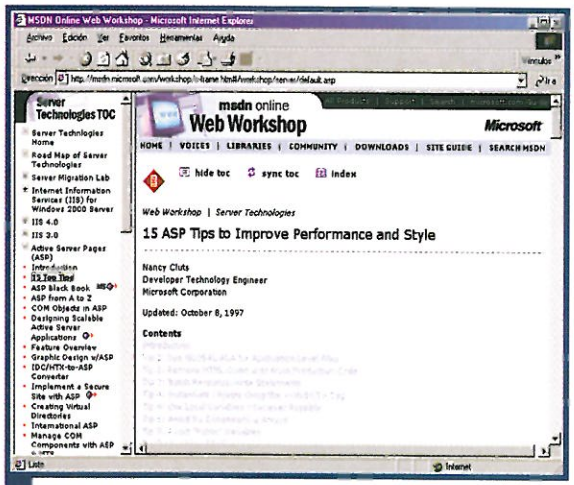


Figura 8.- En *msdn.microsoft.com* podemos encontrar numerosos ejemplos, así como documentación acerca del uso avanzado de páginas *ASP* y formularios.

Criptografía (III)

Javier Sanz Alamillo. Ingeniero de Software.

En este tercer artículo se mostrará un último conjunto de algoritmos de clave privada, entre los que se incluyen *IDEA* y *Blowfish*, detallándose además los tipos de modos de operación en bloque disponibles para éstos.

Los algoritmos de clave privada han sido ampliamente utilizados e implementados en diferentes lenguajes, lo que hace que su utilización en la mayoría de aplicaciones sea muy frecuente. Además, por regla general exigen menor programación y requerimientos, tanto por parte del usuario como del sistema en el que se van a aplicar. Estas razones hacen que el conjunto de algoritmos existentes sea tan amplio y difundido.

LUCIFER

Se trata de un algoritmo que utiliza una clave de 64 *bits* y es capaz de cifrar bloques de 128 *bits* (16 octetos), dividiendo su entrada en dos bloques, *L* y *R*, como ya supondrá el lector. A cada uno de los 8 octetos de la parte *L* se les somete a una transformación no lineal bajo el control de un *bit* determinado por el valor de la clave.

El resultado de dicha transformación se suma mediante una operación "o-exclusiva" con los ocho octetos correspondientes de la clave. Hasta este momento, las transformaciones se realizan al nivel de octeto, pero en los siguientes pasos se realizarán al nivel de *bit*, realizándose transposiciones.

Lucifer sigue siendo el precursor de los algoritmos de clave privada

Una vez realizadas, se agrupan en octetos y se suman *XOR* con los octetos de la parte *R* del bloque de entrada. Entonces, las partes *L* y *R* se intercambian, realizándose 16 iteraciones de este tipo, cuyo resultado es el bloque cifrado.

El descifrado puede realizarse aplicando el mismo proceso al texto cifrado, puesto que se basa en una *red de Feistel.*

IDEA

El algoritmo *IDEA* (*International Data Encryption Algorithm*) es mucho más reciente que el *DES* y tiene mejor consideración general de uso, ya que para la mayoría es el mejor algoritmo simétrico.

Trabaja con bloques de 64 *bits* de longitud y emplea una clave de 128 *bits*. Como el caso del *DES*, se utiliza tanto para cifrar como para descifrar. Como todos los algoritmos simétricos, *IDEA* se basa en los conceptos de difusión y confusión, haciendo uso de las siguientes operaciones:

```
XOR,
suma modulo 216 y
producto modulo 216+1
```

El algoritmo *IDEA* consta de ocho rondas. Se divide el bloque *X* que se va a codificar, de 64 *bits*, en cuatro bloques X_1, X_2, X_3, X_4 de 16 *bits*. Cada uno de estos bloques es la entrada para la siguiente iteración. Se realizan ocho iteraciones, de tal forma que cada bloque X_i sufre un proceso de "o-exclusivo", suma y multiplicación con uno de los otros y con los seis bloques de 16 *bits* en los que se divide la clave. En cada operación el segundo y tercer bloque son intercambiados.

Tras la serie de iteraciones se obtiene el texto cifrado a partir de los cuatro bloques modificados. En la Figura 1 se puede observar gráficamente como se desarrollan todos los procesos descritos anteriormente.

FEAL-N

Su diseño se basa en utilizar una clave de 64 *bits* aplicada sobre bloques de otros 64 *bits*. Este algoritmo es interesante porque ha sido uno de los que ha sufrido los estudios más detallados.

El objetivo planteado al diseñarlo era disponer de un algoritmo similar al *DES* pero que fuera más rápido y seguro. En la Figura 2 puede apreciarse una descripción gráfica de su funcionamiento.

Brevemente, el proceso de cifrado trata un bloque de 64 *bits* de datos (texto en claro o cifrado). A este bloque se le aplica una operación *XOR* con los 64 *bits* que conforman la clave. El bloque se divide en dos. A la parte *L* se le aplica otra operación *XOR* con la parte derecha o *R* y ésta vuelve a ser la parte derecha. Hasta ahora, tampoco es muy original.

Seguidamente, con estos dos bloques se realizan *N* iteraciones (en este caso 4, pero existe también *Feal-8*, por ejemplo). En cada iteración la parte *R* es combinada con 16 *bits* de la clave usando una función *f* y realizando un *XOR* con la parte *L* para conseguir la nueva parte derecha.

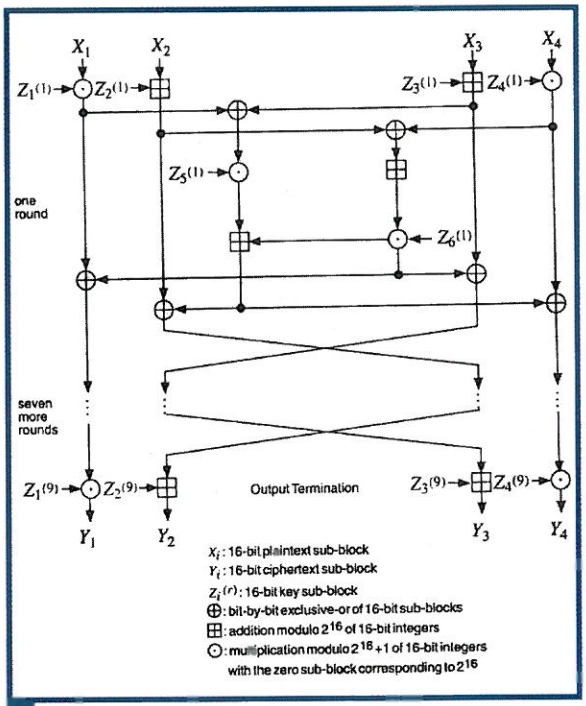


Figura 1.- Imagen del algoritmo *IDEA*.

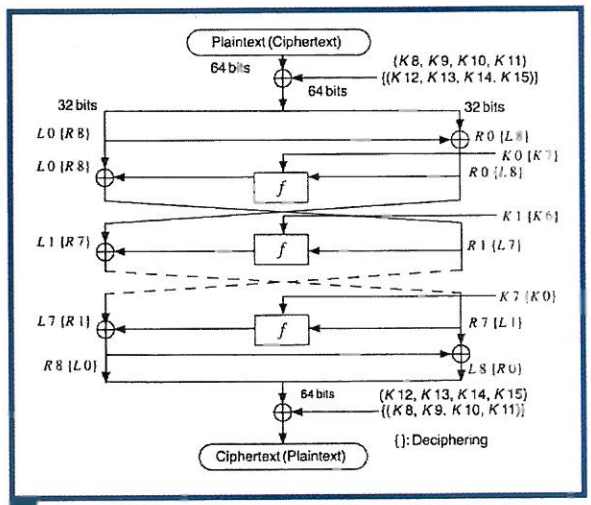


Figura 2.- Imagen del algoritmo *Feal N*.

La parte derecha original (la anterior a las iteraciones) pasa a ser la nueva parte izquierda. Al finalizar las *N* iteraciones, la parte izquierda sufre de nuevo una operación de *XOR* con la parte derecha para crear el bloque de 64 *bits* de datos resultante.

Para terminar, este bloque se modifica con otro *XOR* con otra clave generada de 64 *bits*. En la Figura 3 se muestra este proceso.

Este algoritmo puede utilizarse tanto para cifrar como para descifrar, pero hay que recordar que una vez realizado un cifrado, para que el descifrado concluya correctamente la clave utilizada lo debe ser en orden inverso.

IDEA es un algoritmo bastante extendido, por su rapidez y disponibilidad, en múltiples plataformas y lenguajes

Sin embargo ni *Feal-4* ni *Feal-8* se han demostrado efectivos. Tras estos dos intentos, los diseñadores bautizaron definitivamente el algoritmo como *Feal-N*, indicando así que para que el algoritmo sea seguro se deben realizar *N* iteraciones (siempre superiores a 8).

Finalmente *Feal-N* se modificó y pasó a la historia como el predecesor del nuevo *Feal-NX*, que consta de una clave con 128 *bits*.

NEWDES

Este algoritmo fue diseñado en 1985 por *Robert Scott* como un posible sustituto de *DES*. Realiza operaciones en bloques de 64 *bits* de texto en claro, usando una clave

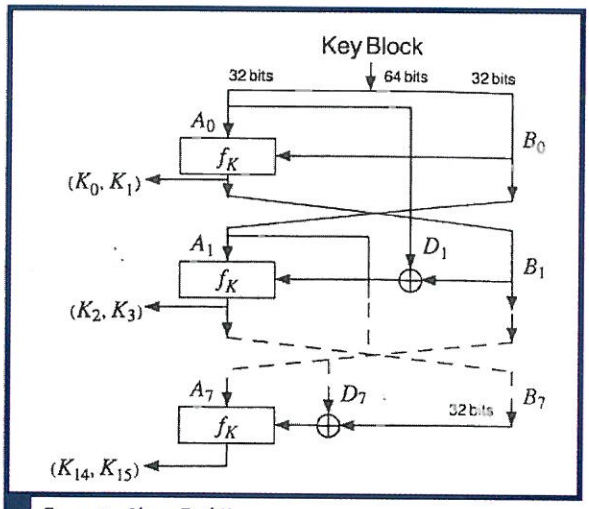


Figura 3.- Clave *Feal N.*

de 120 *bits*. Su funcionamiento es más simple que el *DES*, ya que no hay permutaciones iniciales, y todas las operaciones se realizan sobre octetos, por lo que no se pierde el tiempo tratando *bits* individuales.

Básicamente, su funcionamiento se basa en dividir el texto en claro en bloques de 8 octetos, que se denominan *B0, B1, ... B7*. Entonces se realizan 17 iteraciones. En cada una de éstas, a su vez, se ejecutan 8 pasos. En cada paso, en uno de los bloques se realiza una suma "o-exclusiva" con una parte de la clave, sustituida con otro octeto por medio de una función *f* y "o-exclusiva" con otro bloque.

La clave tiene 120 *bits* de longitud, por lo que se divide en 15 bloques *K0, K1 ... K14*. El proceso puede parecer enrevesado, pero si observa la Figura 4 podrá comprenderlo más fácilmente.

REDOC II

Algoritmo simétrico de cifrado en bloque diseñado por *Michael Wood*. Utiliza una clave de 160 *bits*, en 20 bloques que utiliza para cifrar y descifrar usando cadenas de 80 *bits* de longitud.

NewDES es un buen sustituto de DES

Este algoritmo realiza todas las transformaciones (permutaciones, sustituciones y operaciones *XOR*) sobre octetos, con lo que es bastante eficiente si se utiliza por *software*.

Redoc II utiliza tablas de función *f* variables, en contra del *DES* que las utiliza siempre fijas. En la Figura 5 se puede observar el funcionamiento esquematizado de este algoritmo.

SKIPJACK

Algoritmo desarrollado por la *NSA* utilizando en los *chips Clipper* y *Capstone*. Como la *NSA* continúa clasificando todos sus algoritmos como secretos, no hay mucha información al respecto, aunque esto no garantiza más seguridad.

Las características presentadas sobre el algoritmo públicamente son:

1. Es un algoritmo simétrico.
2. Utiliza una clave de 80 *bits*.
3. Puede ser utilizado en modos *ECB, CFB,OFB* y *CBC*.
4. Se realizan 32 iteraciones para el cifrado y descifrado.
5. La *NSA* comenzó su diseño en 1985 y finalizó su evaluación en 1990.

Debido a que toda la información es secreta, no se puede saber si el algoritmo es seguro o no. Ya que este algoritmo se va a utilizar para el cifrado de los *DMS* (*Defense Messaging System*) se puede deducir que es seguro, pero desde luego, los puristas no están tan convencidos.

BLOWFISH

Creado en 1993 por *Bruce Schneier* con la intención de ofrecer un algoritmo alternativo a los existentes, de forma que fuera rápido y a su vez gratuito. Al igual que con los algoritmos anteriores estamos frente a otro algoritmo que se basa en una red de *Feistel*.

Entre el conjunto de posibilidades de Blowfish se encuentra el permitir el uso de un tamaño variable de clave

Como características principales destacar que aunque sigue utilizando el mismo tamaño de bloque (64 *bits*) como el resto de algoritmos, presenta una serie características que le hacen muy adecuado:

- El tamaño de la clave es impresionante y lo mejor de todo, es que puede ser variable. Se pueden utilizar claves con un tamaño de 32 *bits* hasta 448 *bits*, lo que puede suponer disponer de 156 caracteres de clave!.
- Es totalmente gratuito. Mientras que el resto de algoritmos presentan alguna pega de patentes y similares, este algoritmo es de libre distribución. No se prevé ningún cambio en este sentido.

- Puede usarse en modos *ECB, CBC, CFB, OFB*.
- Es bastante más rápido que *IDEA* o *DES*.

Veamos un ejemplo en el que se utilice *Blowfish*. En los Listados 1 y 2 (este segundo se encuentra en el CD-R) se muestran el cliente y servidor de una aplicación en la que el cliente cifra todos los datos que son enviados a un servidor. Este último, visualiza los datos recibidos previo descifrado.

En el Listado 1 o parte cliente se han introducido algunos cambios para comprobar la versatilidad de desarrollo.

La clase **cli_blowfish** inicialmente lo que hace es generar una instancia de un objeto denominado **conec** que será utilizado en un *thread* aparte del principal.

BlowFish es la mejor alternativa a DES y posiblemente el mejor algoritmo desarrollado

Se ha optado por tener una estructura para la creación de *threads*, ya que por ejemplo, cuando se necesita que varios clientes estén actualizando determinado tipo de información que se almacena en diferentes archivos, con añadir un constructor con el nombre del fichero tendríamos varios clientes enviando a un servidor sin apenas cambios.

La clase **conec** consta de un objeto *Socket* para la conexión con el servidor y en este caso se ha optado por utilizar directamente la clase *BlowfishKey* para realizar las llamadas a *encrypt()*. Como lo que se está realizando es un cliente y el descifrado lo hará el servidor no existirá por tanto llamada a *decrypt()*.

Se siguen necesitando, como es de esperar, los *strings* y tablas de *bytes*. Ahora sólo para cifrar. El método *cifrar()* no varía en su tarea respecto a los anteriores, por lo que pasaremos a *run()* o principal de ejecución en este caso.

En el método *run()*, llamado al realizar la creación y comienzo de ejecución del *thread*, se construye un objeto *BlowfishKey* al que se le pasa una clave (recuerde, como máximo 56 caracteres). En este caso, se ha elegido una longitud fija.

Modifique el programa para que el usuario pueda introducir su clave

LISTADO 1. Cliente Java con BlowFish.

```
class conec implements Runnable
{
  BlowfishKey bfk ;
  String cad_env ;
  String texto_cifrado ;

  byte [] tab_claro;
  byte [] tab_cifrado;

  public String cifrar ( String cadena )
    {
      StringBuffer mensaje_aux = new StringBuffer (cadena);

      int multiplo_8 = mensaje_aux.length() % 8 ;
      for (int cont_car = 0 ; cont_car < (8-multiplo_8);
             cont_car ++) {
            mensaje_aux.append("Z");
        }

      String mensaje = mensaje_aux.toString();
      tab_claro =  mensaje.getBytes();
      tab_cifrado = new byte [ mensaje.length() ] ;

      for (int cont_car = 0 ; cont_car < tab_claro.length ;
            cont_car +=8) {
        bfk.encrypt( tab_claro, cont_car, tab_cifrado , cont_car);
      }

      texto_cifrado = Crypto.hexString(tab_cifrado);
      return texto_cifrado ;
  }

  public void run ()
   {
    bfk = new BlowfishKey ( "sesion".getBytes());
              sock_conex = new Socket("localhost",7000);

    salida = new DataOutputStream (sock_conex.getOutputStream()  );
    teclado = new DataInputStream ( System.in );

    p.println ( "Teclee datos a enviar (fin->salir) : ");
    cad_env = teclado.readLine();
    while ( ! cad_env.equalsIgnoreCase ("salir") )
      {
        texto_cifrado = cifrar ( cad_env );

        p.println ( "Enviando : " );
        p.println ( "Texto en claro : " + cad_env );
        p.println ( "Texto cifrado  : " + texto_cifrado );

        salida.writeBytes ( texto_cifrado + "\r\n" );
        salida.flush();

        p.println ( "Teclee datos a enviar (fin->salir) : ");
        cad_env = teclado.readLine();
      }
   } // run
}
```

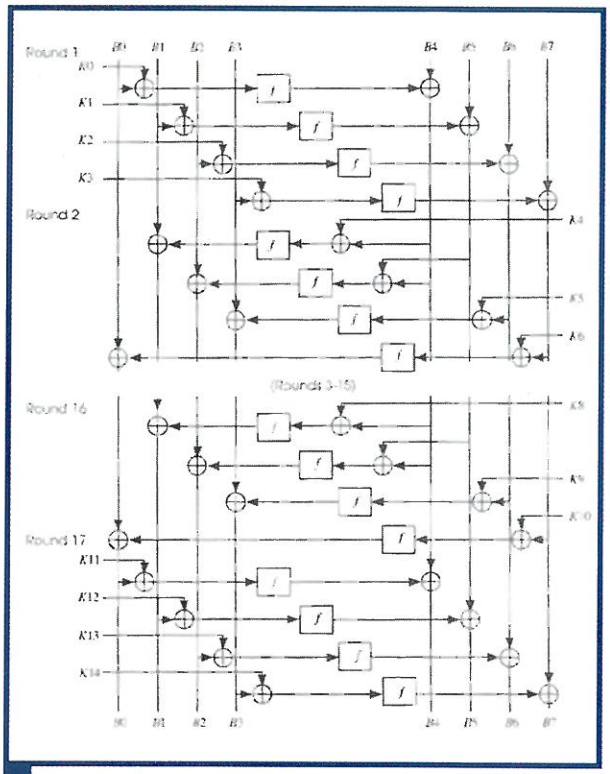


Figura 4.- Algoritmo *NEWDES*.

y no admita claves cuya longitud sea menor de 56 caracteres.

Se construye el *socket* y por tanto la conexión, y los datos que el usuario teclea son cifrados y enviados al servidor. Una vez que se teclea **salir** el programa cierra el *socket*, los descriptores utilizados y finaliza.

El servidor mientras tanto visualizará los datos enviados que han sido descifrados por él mismo. Estudie el código detenidamente. Observe cómo una vez más, no se ha mezclado de forma inseparable las acciones sobre el cifrado de los datos con los de la transmisión de los mismos. Además, puede cambiar el tipo de algoritmo, a por ejemplo *DES*, sólo cambiando el objeto de cifrado que utilizar.

Veamos ahora que pasa en la parte servidor. En el Listado 2, incluido en el *CD-ROM*, puede observar el código *Java* utilizado.

Ahora nos ponemos en el otro lado, ¿qué tiene que hacer el servidor?. Pues recibir los datos del cliente, descifrar y visualizar. En el mismo listado se observa que el servidor, con la clase *serv_blowfish* sólo se encarga de aceptar peticiones mediante la sentencia *accept* en un determinado puerto.

Por cada conexión realizada se construye un *thread* con un objeto tipo conexión en el que se realiza toda la gestión de la comunicación, el descifrado y la visualización.

Lo primero que se hace es crear el objeto *BlowfishKey* con la clave privada. A partir de ese momento, se van leyendo datos del *socket* hasta que se finalice la comunicación. Por cada línea recibida de información cifrada, realizamos el proceso de descifrado como se ha mostrado hasta ahora. Se crean bloques de 8 *bytes* y se descifra.

Como lo que recibimos son caracteres visualizables, no *bytes* puros y duros (se envía el resultado de un *hexString()* directamente leemos sobre un objeto *String)* y realizamos la llamada a *fromHexString()*, con lo que obtenemos un *array* de *bytes* con el resultado del descifrado. Una vez obtenidos todos los bloques se consigue el texto que el cliente envió.

No tiene el servidor más líneas de código que el cliente. Estudie el código detenidamente. Una vez que comprenda las dos partes, modifique tanto el cliente como el servidor para que la comunicación sea bidireccional, de tal forma que, por ejemplo, el servidor se convierta en un clásico servidor de eco, eso sí, con todas las comunicaciones cifradas.

Como se ha visto hasta este momento, la utilización de los métodos criptográficos es simple y sencilla, por lo que tarde o temprano se plantea la duda. ¿Y cuál utilizo comúnmente, cuál es el que mejor se adapta a mis necesidades, el más rápido en mi sistema?.

CUÁL ES EL MEJOR ALGORITMO

La decisión más adecuada está en función de las necesidades, de posibles requerimientos de eficiencia, de experiencias, gustos personales, etc.

Aunque parezca increíble, hoy por hoy *DES* sigue siendo bastante utilizado y relativamente seguro, desde luego no para aplicaciones críticas, pero sí en la mayoría de las aplicaciones que no necesiten mucha seguridad. Existen muchas implementaciones gratuitas y es relativamente rápido. Una opción rápida para resolver un problema dado.

Si desconfía un poco del uso de *DES*, que debería, siempre puede utilizar una variante conocida como *TripleDES* que incrementa la seguridad frente a un ataque por la fuerza bruta, o sea, por el conjunto de todas las posibles claves.

REDOC y *BlowFish* han sido estudiados y no se han publicado

ataques mínimamente efectivos, por lo que pueden ser también buenas opciones, aunque no son muy conocidos y puede que no encuentre alguna implementación que se adapte a sus necesidades. De todos modos, *BlowFish* se presenta como la mejor alternativa "casera" por su velocidad y porque no hay que realizar ningún gasto añadido. Es el más recomendable.

Aunque siempre queda *IDEA*, con su clave de 128 *bits*. Sobre él existen muchos estudios y publicaciones, concluyendo que es un algoritmo seguro, con la tranquilidad de no esperar que aparezca alguien con una *trap-door* (puerta de atrás).

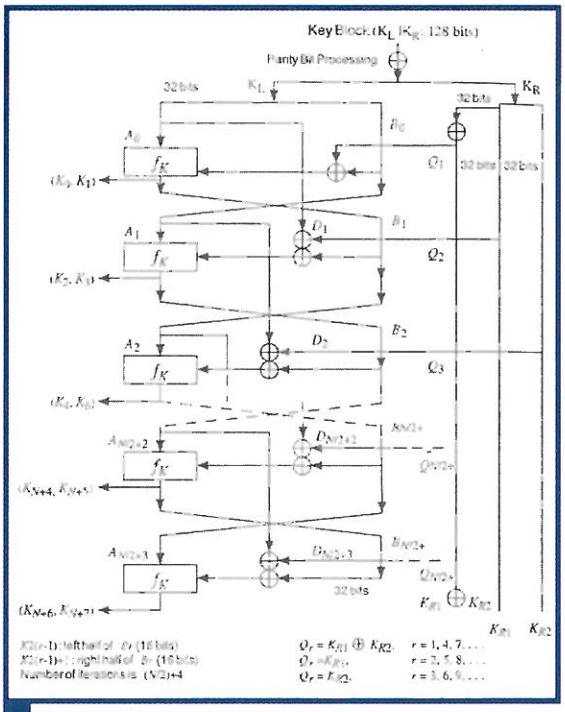


Figura 5.- Algoritmo *REDOC*.

DES sigue siendo muy utilizado en las aplicaciones por el gran número de implementaciones existentes

DES siempre ha tenido mala presentación entre los puristas de la criptografía porque nunca se ha demostrado el porqué de la construcción de las S-Cajas, y por tanto, queda el recelo de que se haya construido para tener una *trap-door* que utilizar.

CIFRADO POR BLOQUES

Como el lector habrá podido observar en los listados presentados, siempre se debe hacer una comprobación de que el tamaño del bloque de texto en claro que cifrar es justo del tamaño a cifrar, es decir, en la mayoría se requieren 8 caracteres de 8 *bits* cada uno, con lo que se obtienen los 64 *bits*, que es el estándar de operación.

Inicialmente esta restricción está bien en caso de que tengamos un volumen de datos que manejar ciertamente reducido, pero nadie espera que al cifrar toda una base de datos haya que ir comprobando por programa si los datos que cifrar son múltiplos de 64 (número de *bits*).

Evidentemente, resolver esta circunstancia en aplicaciones caseras puede basarse en varias posibilidades. Una de ellas ha sido la utilizada en los ejemplos mostrados. Es decir, se rellena el último bloque con un carácter cualquiera, principalmente el carácter blanco. Esto es una solución válida, pero desde luego, pueda dar pistas a un atacante sobre qué tipo de algoritmo estamos usando.

Otra solución que se podría haber aplicado consiste en realizar un relleno o *padding* de los *bytes* que faltan con un patrón y situando en el último de éstos un número que indique cuántos se han añadido, de tal forma que al descifrar, no se obtengan estos últimos, por tanto, no se ofrece información. En la Figura 6 se puede observar gráficamente este proceso de rellenado.

Hasta ahora la idea parece buena, pero tiene una pequeña pega, y es que si el bloque de datos a cifrar es del mismo tamaño, debemos añadir un bloque que contenga exclusivamente los caracteres de relleno, ya que debemos tener una forma de detectar cuándo el bloque se ha rellenado y cuándo no. Una buena solución que implica añadir más caracteres al cifrado.

MODOS DE OPERACIÓN

Al igual que existen diversas formas para realizar el rellenado, también se han definido diversas forma de cifrar los distintos bloques de un texto en claro. Estos medios se definen como modos de operación. Nosotros vamos a ver los más comunes.

MODO ECB

ECB (*Electronic Code Book*) es el método más sencillo y simple a la hora de cifrar bloques. Se ha estado utilizando en los diferentes ejemplos presentados. El proceso se basa simplemente en ir cifrando bloque por bloque utilizando para cada uno de ellos la misma clave.

Como se aprecia, el proceso es sencillo y no implica ningún añadido de tiempo. Además como cada bloque se trata individualmente, es decir, no hay secuencia de orden, no pueden producirse errores que se transmitan de un bloque a otro en los procesos de cifrado y descifrado; los bloques no afectados quedan intactos.

Otra ventaja es que es ideal para aquellas estructuras de datos que no requieran orden de secuencia, por ejemplo, acceso a registros de un fichero aleatorio.

Como desventaja se puede presentar que si alguien modifica un bloque, no hay forma de saber si se ha realizado una modificación o varias, o cuáles son los bloques afectados. También, a igual texto en claro, por ejemplo, se obtendría idéntico texto cifrado, lo que puede dar información a un posible atacante.

MODO CBC

El modo de operación *CBC* (*Cipher Book Chaining Mode*) añade un mecanismo de *feedback* o retroalimentación en el tratamiento de los bloques.

El modo CBC es el más utilizado por sus múltiples ventajas

Esto significa que, por ejemplo, la codificación de los bloques anteriores condiciona la codificación del bloque actual, por lo que el problema de que un bloque puede ser sustituido queda resuelto.

Todo este proceso se realiza simplemente con la aplicación de una operación *XOR* entre el bloque del texto que tratar y el último tratado, del cual se dispone el resultado. En la Figura 7 se puede observar gráficamente este proceso. Puede parecer complicado, pero una revisión detallada hará que lo comprenda claramente.

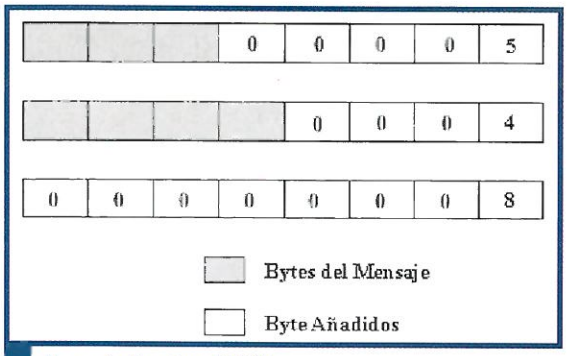


Figura 6.- Algoritmo *REDOC*.

Con esto modo no sólo corregimos una desventaja de *ECB*, sino dos. Con el uso de un vector de inicialización, construido aleatoriamente, se consigue que dos bloques idénticos no ofrezcan el mismo resultado, ahora serán diferentes, ya que el resultado de tratarlo está en función del vector o bloque inicial aleatorio.

Veamos un ejemplo de utilización de este modo de operación. En el Listado 3, que se puede localizar en el *CD*, tenemos la clase **cbc** que realiza el cifrado y descifrado de las líneas que el usuario teclea.

A partir de una clave se realizan estas tareas en el método *pruebaCBC()*. Como siempre hay que controlar el rellenado de los datos que manipular. Una vez realizado esto, vienen los cambios importantes. Con la siguiente sentencia:

```
EncryptCBC c_cbc = new EncryptCBC
    (dk );
```

se consigue obtener un objeto que realiza el cifrado en modo de operación *CBC*, con la ventaja de que él mismo se encarga de realizar todos los cifrados correspondientes al conjunto de bloques que sean necesarios.

El conjunto y tamaño del bloque es obtenido ya que se le pasa al constructor un objeto del tipo *dk* que es un objeto *DesKey()*. Con ello se invoca al método *encrypt()* que realiza toda esa tarea y devuelve un *array* de *bytes* con el texto cifrado.

```
c_cbc.encrypt( tab_claro, 0,
    tab_claro.length );
```

El proceso de descifrar pasa por el camino contrario. Se crea un objeto del tipo *DecryptCBC* como se muestra en la siguiente línea:

```
DecryptCBC d_cbc = new DecryptCBC
    (dk);
```

Una vez realizado esto se invoca al método *decrypt()* con la referencia *d_cbc* y se obtiene un *array* de *bytes* con el resultado de descifrar. Se pasa a un objeto *String* y se visualiza. Como se observa, el tratamiento de la información con este modo es muy cómodo y eficaz.

MODO CFB

Tanto en *ECB* como en *CBC* no se empieza a tratar el bloque de datos hasta que éste está completo. Esto puede que siempre se cumpla en determinadas condiciones, pero existen muchas aplicaciones en las que no se puede esperar a realizar este proceso hasta que se llena el bloque, sino que se debe realizar por cada unidad de tratamiento que se gestiona.

Por ejemplo, si estamos cifrando la comunicación de dos terminales orientados a carácter que trabajan en modo de comunicación *full-duplex*, no podemos esperar a que se rellene todo un bloque de datos, tratarlo y enviarlo. Para remediar esto, hay que realizar un relleno como anteriormente se ha mencionado o utilizar el modo *CFB*.

El modo de operación *CFB* (*Cipher Feedback Mode*) permite tratar los datos en unidades inferiores al tamaño del bloque utilizado, manteniendo un nivel de seguridad adecuado.

Al igual que en *CBC*, un error en la transmisión de los datos tratados implica que el resto de la comunicación queda invalidada. En algunos casos esto es interesante, puesto que garantiza que no se han alterado los datos, pero en otros casos, como la transmisión de datos, puede resultar algo problemático.

MODO OFB

Hay circunstancias en las que el cifrado de bloques presenta ciertas molestias que, llegados a determinados extremos, hacen poco práctica su utilización.

Existen procedimientos criptográficos que no se basan en la gestión de bloques, sino como se detalló en el primer artículo en el cifrado *Vernam*, el tratamiento de los datos se basa en secuencias de bits o *streams* de tamaño variable.

Estos algoritmos permiten el tratamiento por *bits* y su gestión hasta que se requiera gestionar el siguiente *b-it*. Funcionan a partir de lo que se denomina *generador de secuencia de clave*, y usando los *bits* generados por éste y los propios de los datos que tratar obtenemos un resultado que es gestionado como sea necesario. El proceso inverso requiere de una sincronización en el generador de la secuencia de clave, lo cual es un pequeño inconveniente.

El modo de operación *OFB* (*Output Feed Back*) puede ser utilizado para generar una secuencia de clave, y similar en funcionamiento al modo *CBC* salvo que se realizan una serie de desplazamientos en los datos y se utiliza una retroalimentación interna añadida para garantizar la independencia de los datos que se puedan cifrar en un *stream*.

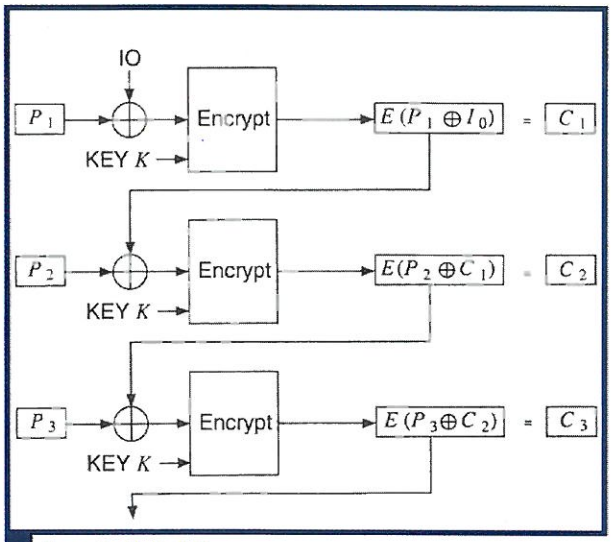


Figura 7.- Modo *CBC*.

Aunque este modo de operación se puede utilizar con bloques de datos inferiores al tamaño de su gestión, se ha demostrado que únicamente se debería utilizar cuando el tamaño de los datos coincide con el tamaño del bloque a manipular.

En el Listado 4, localizable en el *CD-ROM*, se muestra un ejemplo de utilización de *OFB* usando *Cryptonite*. En él se utiliza la misma filosofía que en el anterior ejemplo, sólo hay que indicar cuál es el tamaño del bloque a cifrar.

Con OFB se puede definir el tamaño del bloque que se va a tratar

En la siguiente sentencia se observa cómo en el constructor se indica la clave a utilizar y el tamaño del bloque sobre el que se aplicará.

```
EncryptOFB c_ofb = new EncryptOFB
    (dk, 1 );
```

Al igual que con *CBC*, con invocar el método *encrypt()* se realiza el cifrado de los datos, y no hay necesidad de realizar ningún bloque repetitivo. Lo interesante de *OFB* es que como se ha definido el tamaño de bloque a 1, no hace falta realizar ningún tipo de *padding* sobre los datos.

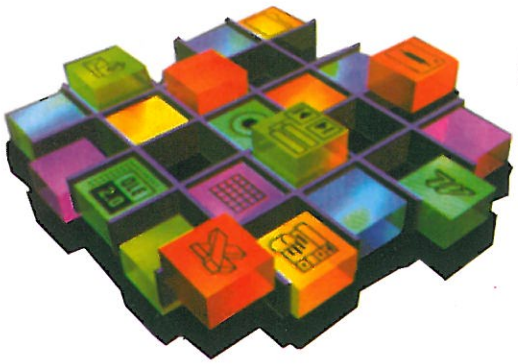
El proceso de descifrado se basa en el uso de objetos tipo *DecryptOFB()* con los cuales se invocan a los métodos *decrypt()*.

Revise el código detenidamente. Gracias a la librería que se está utilizando, todas estas tareas son sumamente fáciles y prácticas.

CONCLUSIÓN

Se ha presentado y descrito el conjunto de criptosistemas privados más comunes. Con los ejemplos desarrollados, el lector puede comprobar cómo utilizar los algoritmos en sus aplicaciones de forma rápida, eficaz y sencilla. Además, mediante el uso de los modos de operación se pueden ofrecer ciertas garantías de seguridad, evitando así posibles ataques a la información.

En el próximo artículo se mostrarán los criptosistemas de clave pública así como las funciones para utilizar firma digital.



Suite para Internet (IV): Creación de un chat

Jordi Agost Moré.
Profesor de Programación/Multimedia de la Universidad de Lleida.

¿Cómo podemos añadir una comunicación de tipo *chat* a nuestros programas? ¿Cuáles son las aplicaciones que puede tener dicha comunicación sobre la suite que estamos desarrollando? ¿Cuáles son los pasos que debemos seguir? ¿Cómo lo podemos realizar con el menor número de controles posibles? A éstas y otras preguntas intentaremos responder a través de este artículo.

INTERNET Y EL CHAT

Uno de los usos más populares de *Internet* lo constituye la posibilidad de establecer una conversación escrita o hablada con otras personas. Es lo que conocemos como *IRC* (*Internet Relay Chat*) o simplemente *chat*.

Dentro de los *chat*s las personas buscan todo tipo de relaciones con otras personas: desde una comunicación de tipo profesional, hasta compartir experiencias, intereses comunes o simplemente amistad. Vamos a ver en este artículo cómo realizar una aplicación básica de este tipo. Dicha aplicación estará constituida a su vez por un par de aplicaciones independientes: una de tipo servidor y otro de tipo cliente, para que podamos gestionar correctamente el *chat*.

Normalmente el usuario utilizará la aplicación cliente para conectarse a un servidor de *chat*, con el fin de poder "hablar" con otras personas.

La aplicación de tipo servidor sería la que se instalaría en el servidor de *Internet* y a la que se conectarían diversas personas para hablar entre sí, mediante el uso de la citada aplicación cliente. En primer lugar veremos la construcción

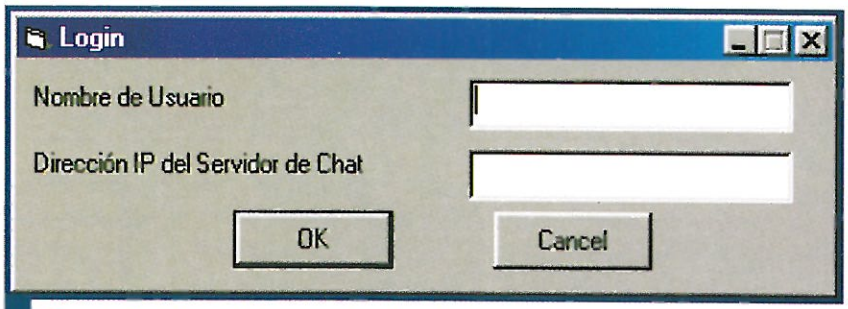


Figura 1.- Pantalla inicial de nuestra la aplicación cliente del *chat*.

de una aplicación cliente, para, posteriormente seguir con la creación de la aplicación servidor.

APLICACIÓN CLIENTE DEL CHAT

PANTALLA INICIAL

Vamos a dividir nuestra aplicación cliente en dos formularios. El primero se utilizará para pedir al usuario los datos que se necesitan para empezar a trabajar con nuestra aplicación de *chat*. Dichos datos son por una parte el nombre con el cual nos identificaremos y por otra la dirección *IP* de la aplicación servidor de *chat*.

Al pulsar el botón **Aceptar** de dicho formulario se ejecutará la que vamos ahora a analizar con más detalle a continuación:

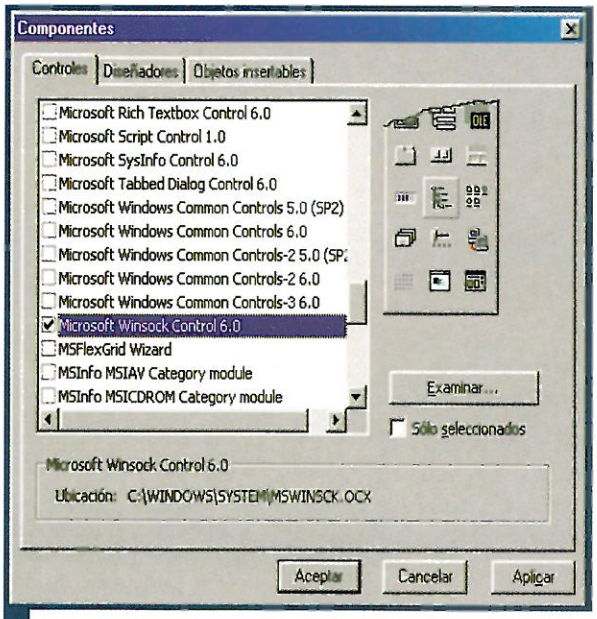


Figura 3.- Deberemos activar las referencias al control *Winsock* para poderlo utilizar en nuestro proyecto.

```
Private Sub cmdOK_Click()

On Error GoTo CtrolErrores:

frmLogin.Caption = "Entrada -
    Conectando..."
frmMain.wskClient.Connect
    txtIPServidor.Text, 1000
Exit Sub

CtrolErrores:
MsgBox "La dirección IP no es
    válida!", vbExclamation,
    "Atención"
frmLogn.Caption = "Entrada"
frmMain.wskCliente.Close
End Sub
```

Winsock será el encargado de la comunicación en nuestra aplicación

En ella lo primero que hacemos es poner en el título del formulario la expresión **Conectando....** A continuación invocaremos el método *Connect* del control *Winsock* presente en el formulario principal pasándole como argumento la dirección *IP* que se encuentra en la caja de texto, y a través del puerto **1000**.

Si durante la ejecución de la rutina se ha producido algún error el flujo de programa se dirigirá a la etiqueta *CtrolErrores* en la que se informará que la dirección *IP* del servidor no es válida y se cerrará la comunicación del control *Winsock* del for-mulario principal: **frmMain**.

Figura 2.- El control en la barra de herramientas una vez activada la referencia.

También en este primer formulario establecemos el nombre de usuario. Así, cada vez que el contenido de la caja de texto de nombre de usuario cambie, actualizaremos una variable global llamada **strNombreUsuario**, que se encuentra declarada dentro del módulo *modPpal*:

```
Private Sub
 txtNombreUsuario_ Change()

strNombreUsuario =
 txtNombreUsuario.Text

End Sub
```

MÓDULO PRINCIPAL

Una vez realizado el formulario de petición de datos o *Login* entraremos en la pantalla que constituye el núcleo principal de nuestra aplicación cliente. Fijémonos en que dicha aplicación posee un control *Winsock* que será el encargado de la comunicación, y para que sea visible en nuestra barra de herramientas deberemos activar previamente una referencia para el control (Menú **Proyecto+Referencias**).

La primera rutina que debemos analizar es la que se corresponde

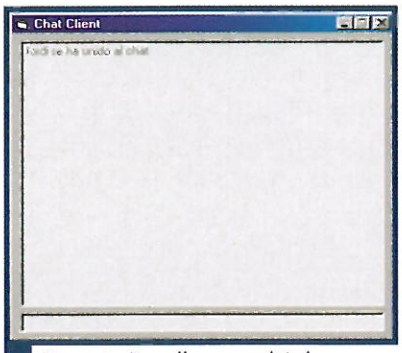


Figura 4.- Pantalla principal de la aplicación cliente.

con el evento *Connect* del control *Winsock*. Dicha rutina se ejecuta cuando el control *Winsock* ha establecido una comunicación y será la encargada de mostrar el formulario principal y de descargar el formulario de entrada o de *Login*.

```
Private Sub wskCliente_Connect()

wskCliente.SendData "Nombre" &
     vbTab &
 strNombreUsuario & vbCrLf
frmPpal.Show
Unload frmLogin

End Sub
```

En el supuesto de que el control *Winsock* no pueda establecer la comunicación con la aplicación servidor de *chat* se producirá un error con lo que se ejecutará el método *Error*, que será el encargado de mostrar el error y descargar el formulario principal, volviendo a mostrar el formulario de entrada.

```
Private Sub
     wskCliente_Error(ByVal Number
     As Integer,
 Description As String, ByVal
     Scode As Long,
 ByVal Source As String, ByVal
     HelpFile As String,
 ByVal HelpContext As Long,
     CancelDisplay As Boolean)

MsgBox "Error: " & Number & " - "
     & Description
frmLogin.Show
Unload frmPpal

End Sub
```

Pero donde se produce realmente todo el proceso del cliente es en el evento *DataArrival*. En dicha rutina lo que hacemos es coger lo que nos enviará la aplicación servidor y ponerlo en la variable *strDatos*, posteriormente analizamos lo que hemos recibido, y dependiendo de la palabra clave realizaremos una acción u otra.

Todo el proceso del cliente se produce en el puerto DataArrival

Si la palabra clave es **MENSAJE**, simplemente mostraremos los datos por pantalla con el fin de que el usuario pueda conocer y seguir la conversación. Si la palabra que recibimos es **KICK** entonces le comentaremos al usuario que ha sido expulsado del servidor de *chat*s y lo devolveremos a la pantalla de inicio, para que vuelva a intentar entrar en este o en otro servidor.

```
Private Sub
     wskCliente_DataArrival(ByVal
     bytesTotal As Long)

Dim intx
Dim strDatos As String,
     strDatosPpal() As String,
 strDatosOrdenados() As String
wskCliente.GetData strDatos,
     vbString
strDatosPpal = Split(strDatos,
     vbCrLf)
For intx = LBound(strDatosPpal)
     To UBound(strDatosPpal) - 1
 strDatosOrdenados =
     Split(strDatosPpal(x), vbTab)
 Select Case strDatosOrdenados(0)
 Case "MENSAJE"
 txtConversacion.SelStart =
     Len(txtConversacion.Text)
 txtConversacion.SelText =
     strDatosOrdenados(1) & vbCrLf
 Case "KICK"
 MsgBox "Te han hechado del ser-
     vidor por la siguiete razón:
     " &
 strDatosOrdenados(1),
     vbInformation, "KICK:"
 frmLogin.Show
 Unload Me
 End Select
Next intx

End Sub
```

Cada vez que el usuario pulsa una tecla miraremos si la que ha pulsado es la tecla de "retorno de carro" o *Enter*, ya que en este caso enviaremos la información que el usuario hubiese escrito hasta ese momento.

```
Private Sub
     txtEntrada_KeyDown(KeyCode As
     Integer, Shift As Integer)

If KeyCode = vbKeyReturn Then
 wskCliente.SendData "Message" &
     vbTab &
 strNombreUsuario & ": " &
     txtEntrada.Text & vbCrLf
txtEntrada.Text = ""
End If

End Sub
```

APLICACIÓN SERVIDOR DEL CHAT

CARGANDO LA APLICACIÓN

Cuando carguemos la aplicación servidor por primera vez, se eje-

cutará, como es lógico, el evento *Form_Load* del formulario:

```
Private Sub Form_Load()

On Error Resume Next
Dim intx As Integer

For intx = 0 To CSTNUMMAXSOCK
 Load wskServidor(x)
 typUsuariosConectados(x).Libre =
    True
Next intx

wskEscucha.LocalPort = 1000
wskEscucha.Listen
lblIPServidor.Caption = "SERVIDOR
    - " & wskEscucha.LocalIP

End Sub
```

En dicho evento lo primero que haremos es cargar tantos controles como indique la constante *CSTNUMMAXSOCK*. Para nuestro caso en particular hemos establecido dicho valor en **49** controles, lo que nos da un total de 50 conversaciones posibles (recordemos que empezamos a contar desde **0**).

Al mismo tiempo en que los cargamos inicializamos la estructura de los usuarios. Seguidamente establecemos el puerto de escucha con el valor de **1000** (recordemos que la aplicación cliente se conecta por el mismo puerto) y en la etiqueta del servidor ponemos la dirección *IP*. A continuación invocamos el método *Listen* para así escuchar a posibles usuarios que deseen conectarse.

ACEPTANDO UNA COMUNICACIÓN

Cuando el control *Winsock wskEscucha* recibe una cierta petición a través del puerto número **1000** se activará el evento *ConnectionRequest*. A partir de este momento buscaremos si en la estructura de datos de los usuarios hay algún "hueco", es decir, algún elemento que en ese momento no se encuentre comunicándose con un cliente. Si se trata de este caso lo que haremos es asignarle la conversación entrante, tal y como se muestra en la siguiente rutina:

```
Private Sub
    wskEscucha_ConnectionRequest(
 ByVal requestID As Long)

Dim intx As Integer
For intx = 0 To CSTMAXNUMSOCKETS
 If typUsuarios(x).Libre = True
    Then
 typUsuarios(x).Libre = False
 wskServidor(x).Accept requestID
Exit For
End If
Next intx

End Sub
```

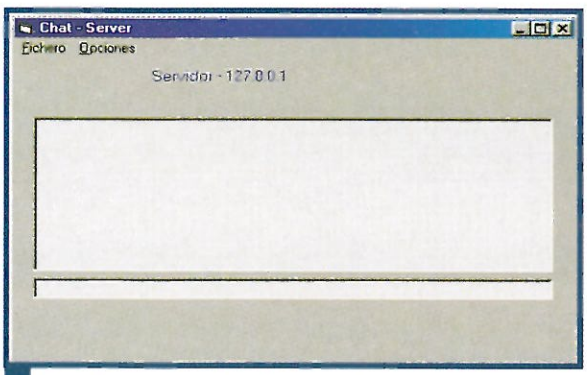


Figura 5.- Aspecto de la aplicación del servidor.

PROCESANDO ENVÍOS

Al igual que en la aplicación cliente, la aplicación servidor dispone de una rutina que procesa la información que le llega de la otra aplicación con la que está en comunicación. Dicha rutina es muy similar y al igual que en la aplicación cliente elige el tipo de mensaje que le llega y decide mostrarlo por pantalla si es un mensaje simple o unir una persona al *chat* si el mensaje contiene la palabra **NOMBRE**. Veamos a continuación el código utilizado:

```
wskServidor(Index).GetData Data,
    vbString
MainData = Split(Data, vbCrLf)
For x = LBound(MainData) To
    UBound(MainData) - 1
 SplitData = Split(MainData(x),
    vbTab)
 Select Case SplitData(0)
 Case "MENSAJE"
 EnviarMensaje SplitData(1)
 Case "NOMBRE"
 typUsuarios(Index).Nombres =
    SplitData(1)
 EnviarMensaje
    typUsuarios(Index).Nombres &
    " se ha unido al chat"
 End Select
Next x
```

ENVIANDO MENSAJES

Cuando en el evento *KeyDown* se pulsa la tecla **Enter**, tal y como se ve a continuación:

```
If KeyCode = vbKeyReturn Then
 EnviarMensaje "MENSAJE DEL SER-
    VIDOR:" & txtEntrada.Text
 txtEntrada.Text = ""
End If
```

El envío de mensajes en conversación activa se realiza con el método SendData

El contenido de lo que el usuario había escrito previamente es enviado a las demás aplicaciones con las que hay una conversación. Esto se efectúa mediante la rutina *EnviarMensaje*.

En dicha rutina lo primero que hacemos es mostrar por pantalla el mensaje que vamos a enviar. A continuación recorreremos el número total de controles *Winsock* que tengamos e iremos comprobando uno a uno los que tienen una conversación activa. A los que la posean les enviaremos el mensaje mediante el método *SendData* del control *Winsock* correspondiente. Veamos a continuación el código asociado:

```
Private Sub
     EnviarMensaje(strMensaje As
     String)

Dim intx

txtSalida.SelStart =
     Len(txtSalida.Text)
txtSalida.SelText = strMensaje &
     vbCrLf

For intx = 0 To CSTMAXNUMSOCKETS
 If typUsuarios(x).Libre = False
     Then
 wskServidor(x).SendData "MENSA-
     JE" & vbTab & strMensaje &
     vbCrLf
 DoEvents
Next intx

End Sub
```

CERRANDO UNA COMUNICACIÓN

En el caso de que un cliente decida cerrar la conversación con nuestra aplicación servidor se producirá el evento *Close* del control *Winsock* correspondiente (recordemos que estamos en un *array* de controles). En dicho caso lo primero que deberemos hacer es limpiar la estructura de datos correspondiente a la comunicación en cuestión, cerrar la comunicación del control y notificarlo al usuario tal y como vemos a continuación:

```
Private Sub
     wskServidor_Close(Index As
     Integer)
 typUsuariosConectados(Index).Name
     = ""

     typUsuariosConectados(Index).
     Libre = True
 wskServidor(Index).Close
 EnviarMensaje
     typUsuariosConectados(Index).
     Name &
 " ha abandonado la sesión de
     chat."
End Sub
```

FINALIZANDO LA APLICACIÓN

En el supuesto de que la aplicación que estamos desarrollando termine o se descargue de la memoria, lo que haremos es recorrer el número total de controles *Winsock* que hemos cargado y los iremos descargando uno a uno, para así finalizar todas las conversaciones que tengamos activas:

```
Private Sub Form_Terminate()

On Error Resume Next
For x = 1 To 49
 Unload wskServidor(x)
Next x

End Sub
```

CONCLUSIÓN

En este artículo hemos visto cómo podemos crear una comunicación de *chat* dentro de nuestras aplicaciones, tanto desde el punto de vista del cliente como del servidor. A este tipo de aplicaciones podríamos dedicarle una finalidad simplemente lúdica como ocurre algunas veces con los *chats*, pero, por ejemplo, también podemos aprovecharlas para finalidades profesionales.

Cuando se cierra una comunicación hay que limpiar la estructura de datos

De hecho podemos incorporar un cliente *chat* a nuestros programas para que los usuarios se comuniquen entre sí o bien puedan llegar a establecer contacto con los creadores de los mismos y realizar intercambios de ideas. De esta forma, podremos, a partir de este momento, aplicar los conocimientos adquiridos para dotar de una mayor funcionalidad a nuestras aplicaciones.

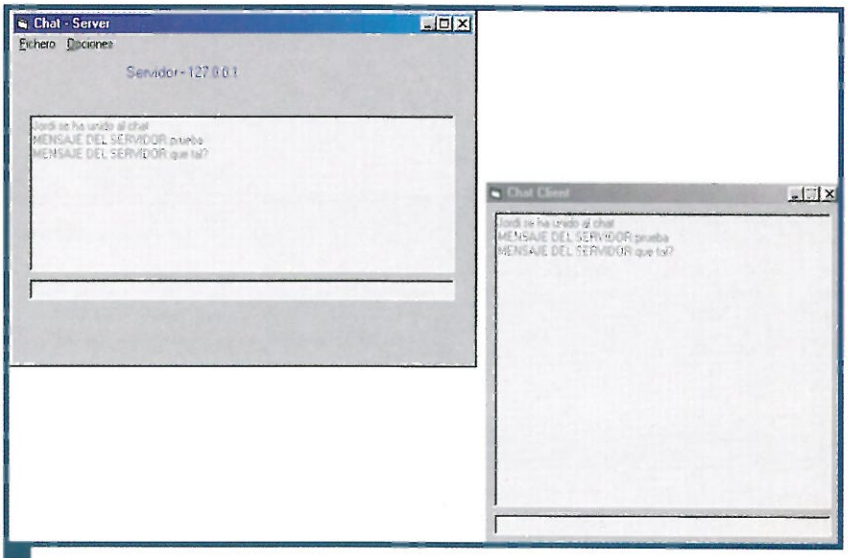


Figura 6.- La comunicación desarrollada entre las dos aplicaciones creadas.

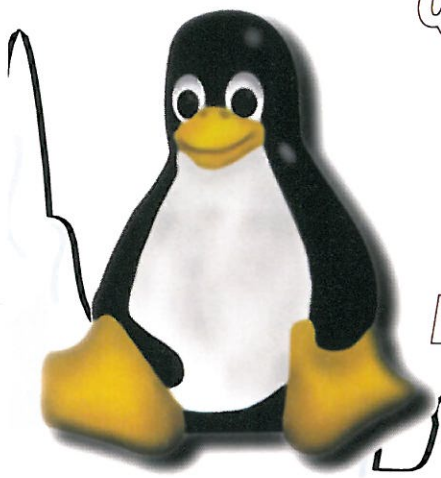
SUSCRIPCIÓN DOBLE

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES

Sólo PROGRAMADORES LINUX

QUE NO TE LÍEN CON EL <CÓDIGO>



PARA NO QUEDARSE HELADO CUANDO HABLEN DE LINUX

BOLETÍN DE SUSCRIPCIÓN

Rellene o fotocopie el cupón y envíelo a REVISTAS PROFESIONALES, S.L.
C/ San Sotero, 5. 1ª Planta. 28037 Madrid. Tlf: 91 304 87 64. Fax: 91 327 13 03

Quiero suscribirme a la revistas SÓLO PROGRAMADORES y SÓLO PROGRAMADORES LINUX desde el Nº.....y beneficiarme de las condiciones de estas magníficas promociones:

❑ **SUSCRIPCIÓN ANUAL**
24 NÚMEROS + 24 CD-ROMs
AL PRECIO DE
14.785 ptas. / 88,86 €

❑ **SUSCRIPCIÓN ANUAL ESPECIAL ESTUDIANTES**
24 NÚMEROS + 24 CD-ROMs
por sólo 11.830 ptas. / 71,01 €

FORMAS DE PAGO:

❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Transferencia al Banco Popular Español. C/ Valdecanillas, 41.
Nº c/c: 0075/1040/43/ 0600047439
❑ Talón bancario a nombre de REVISTAS PROFESIONALES, S.L
❑ Domiciliación bancaria
❑ Contra reembolso

Soy antiguo suscriptor
❑ **Sí** ❑ **No**

PARA ENVÍOS AL EXTRANJERO SÓLO SE ADMITIRÁN LAS SIGUIENTES FORMAS DE PAGO:

❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Transferencia al Banco Popular Español. C/ Valdecanillas, 41.
Nº c/c: 0075/1040/43/ 0600047439
❑ Eurocheque conformado con un banco español a nombre de REVISTAS PROFESIONALES, S.L

NOMBRE Y APELLIDOS:.....

EDAD:..... PROFESIÓN: .....

TFNO: ..... DOMICILIO: .....

CIUDAD:..... C.P: .....PROVINCIA:.....

Promoción válida hasta agotar existencias

Datos de domiciliación:

Banco.....

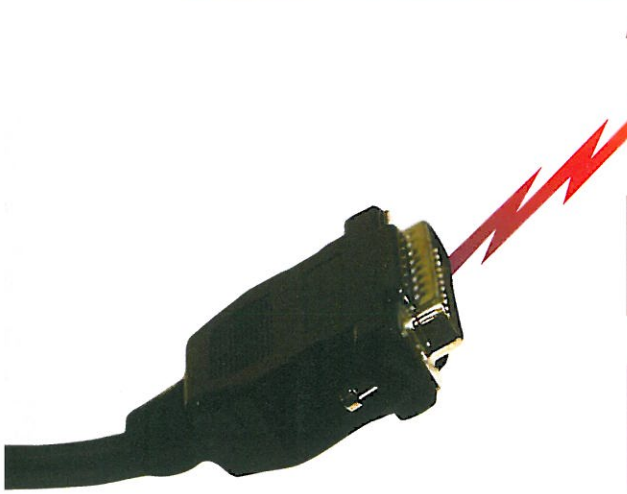
Domicilio .....

Nº de Cuenta.....

Titular.....

Fecha .....

FIRMA



TRANSFERENCIA DE ARCHIVOS PUNTO A PUNTO (V)

Enrique de la Lastra. Desarrollador Independiente.

Después de haber definido y delimitado las pautas de funcionamiento y sus posibles excepciones, en este artículo y en el próximo terminaremos de programar un componente *Delphi* que las desarrolle. Será para un protocolo de nivel de enlace.

INTRODUCCIÓN

El artículo anterior sirvió como introducción a las funciones que debe cumplir un componente que implemente un protocolo de nivel de enlace. Además, en él definimos el comportamiento de nuestro propio protocolo, delimitando las funciones de las que se va a hacer responsable, definiendo el formato de las tramas y dibujando gráficamente el diagrama de una comunicación de datos, en el nivel de enlace, entre dos puntos directamente conectados.

En este artículo crearemos los tipos, variables y procedimientos necesarios para la implementación del componente *Delphi* que va a llevar a cabo las funciones del protocolo de nivel de enlace que hemos inventado.

TIPOS DE TRAMAS

Del artículo anterior nos quedaba pendiente definir las tramas concretas que íbamos a emplear. Como las tres tramas posibles son: trama de información, trama de *ACK* y trama de *NACK*, habrá que definir un código para cada una de ellas. Teniendo en cuenta que la tabla *ASCII* ya define un código para el *ACK* y otro para el *NACK*, emplearemos estos códigos como identificador del tipo de estas tramas. En cuanto a la trama de información, emplearemos el carácter *ASCII* de la letra *"I"* (de información).

En nuestro componente definimos unas constantes, por si en un futuro queremos cambiar estos códigos, para que la reprogramación sea más sencilla:

```
const
  TRAMA_ACK  = $06 {ACK en
    ASCII};
  TRAMA_NACK = $15 {NAK en
    ASCII};
  TRAMA_INFO = $49 {'I' en
    ASCII};
```

Y por tanto, las tramas tendrían el formato presentado en la Figura 1

Además, definimos otras constantes con el fin de continuar cumpliendo el mismo criterio: que en caso de cambio del protocolo nos cueste poco trabajo cambiar el código. Estas constantes serán las

siguientes: primero una constante que almacene el carácter que ejerce la función de *flag*.

```
const
{ Byte que indica el principio de
    la trama }
 PRINCIPIO_TRAMA = $7D
    {01111110};
```

Otras constantes harán referencia al tamaño de cada uno de los campos de la trama, así como a su tamaño total.

```
const
{ Número de bytes de cada campo
    de control dentro de la trama
    }
  NUM_BYTES_PRINCIPIO_TRAMA = 1;
  NUM_BYTES_TIPO_TRAMA      = 1;
  NUM_BYTES_SECUENCIA_TRAMA = 1;
  NUM_BYTES_LONGITUD_TRAMA  = 2;
  NUM_BYTES_CRC_TRAMA       = 2;
{Número de bytes totales de una
    trama, sin contar con el
    campo de información}
  NUM_BYTES_TRAMA  =
    NUM_BYTES_PRINCIPIO_TRAMA +
    NUM_BYTES_TIPO_TRAMA      +
    NUM_BYTES_SECUENCIA_TRAMA +
    NUM_BYTES_LONGITUD_TRAMA  +
    NUM_BYTES_CRC_TRAMA;
```

Otras dos constantes guardarán el tiempo, en milisegundos, tras el cual vence el *timeout*, y el tiempo máximo con que vamos a fijar el *timeout* variable, que utilizaremos en el caso de que los dos extremos de la comunicación transmitan en el mismo instante.

```
const
  TIMEOUT_TX = 500;
  TIMEOUT_TX_VARIABLE =
          2 * TIMEOUT_TX;
```

El *timeout* se define con un valor que sea mayor que dos veces el tiempo que tarda una trama en viajar desde el transmisor al receptor. La razón es simple: si el valor del *timeout* fuera menor que ese tiempo estaríamos repitiendo la misma trama hasta el infinito (Figura 2).

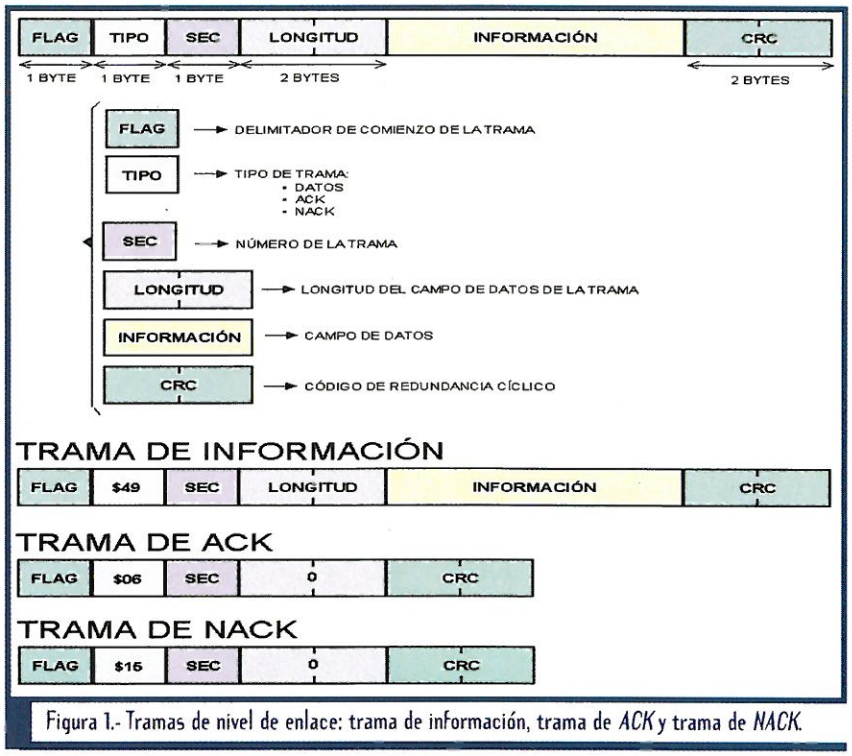


Figura 1.- Tramas de nivel de enlace: trama de información, trama de *ACK* y trama de *NACK*.

Por cierto, dado que el proceso de reenvío de tramas es automático, para no enviar tramas de forma indefinida a un receptor que puede estar apagado, fijaremos otra constante con un valor que indique el máximo número de veces que vamos a repetir la transmisión. En un principio asumiremos un valor máximo de diez repeticiones.

```
const
  NUM_MAXIMO_REPETICIONES = 10;
```

LLEGADA DE INFORMACIÓN ÚTIL

Para que sea lo más simple posible el uso del componente, se debe programar un evento que avise de la llegada de datos, de forma que el nivel superior pueda realizar el tratamiento de la información desde el manejador del evento. El evento llevará, además del aviso implícito de la llegada de información al puerto, un puntero a la estructura de datos útiles que se haya recibido.

De esta manera cuando se detecta que ha llegado información al puerto, se recoge ésta en la estructura de datos *TramaRx*. Si la información recibida es correcta (es decir, no contiene *bits* erróneos) se genera un evento en el que el único parámetro sea un puntero a esa estructura de datos.

Teniendo en cuenta que el evento que genera el componente *ComSerie* que hemos creado en los primeros artículos de la serie es:

```
TOnDataReceived = procedure
  (Sender: TObject;
  Datos: pointer;
  NumDatos: integer) of object;
```

Creamos ahora un evento que avise de la llegada de información útil. Además, como esta información queremos tratarla de forma

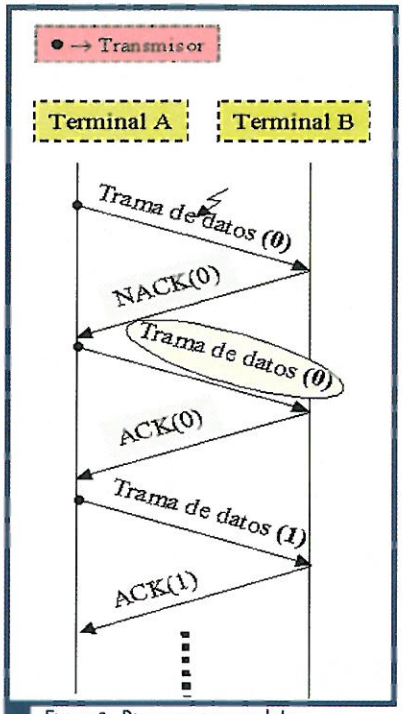


Figura 2.- Diagrama temporal de representación del funcionamiento del protocolo.

mucho más sencilla, el parámetro que apunte a la información útil será un *PChar* (en lugar de un *pointer como sucedía antes*). Su sintaxis sería la siguiente:

```
TOnInfoReceived = procedure
  (Sender: TObject;
  Datos: PChar) of object;
```

Cuando lleguen datos, se disparará el evento *OnDataReceived*. Redefiniremos este evento en nuestra clase *TComEnlace* para poder procesar la información que lleva en sus parámetros y al mismo tiempo, que pueda ser utilizada por todas las clases hijas.

El evento heredado *OnDataReceived* lo asociaremos a un procedimiento que denominaremos *RecepcionDatos*. En este procedimiento se procesará la información recibida, se separarán las tramas, se comprobará si las recibidas son las tramas esperadas y cuando lleguen con información útil, se entregarán al nivel superior a través del evento *OnInfoReceived*.

Listado 1: Definición y utilización de los eventos OnDataReceived y OnInfoReceived.

```
TComEnlace = class (TComSerie)
private
  FOnInfoReceived: TOnInfoReceived;
...
private
  // procedimiento que procesa los datos recibidos en el puerto serie:
  // será el procedimiento que se llame cuando se dispare el evento OnDataReceived
  procedure RecepcionDatos (Sender: TObject; Datos: pointer; NumDatos: integer);

published
  // redefinimos el evento OnDataReceived, que se dispara cuando
  // han llegado bytes al puerto serie
  property OnDataReceived: TOnDataReceived
      read FOnDataReceived write FOnDataReceived;

  // definimos el evento OnInfoReceived, que se disparará sólo
  // cuando hayan llegado datos útiles, y además éstos sean correctos
  property OnInfoReceived: TOnInfoReceived
     read FOnInfoReceived write FOnInfoReceived;
end; { TComEnlace }

procedure Register;

implementation

constructor TComEnlace.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
...
  // Añadimos inherited a OnDataReceived, con lo cual inicializamos
  // el evento OnDataReceived heredado en lugar del que hemos redefinido
  inherited OnDataReceived := RecepcionDatos;

  // Inicializamos a nil las variables de nuestros nuevos eventos
  FOnDataReceived := nil;
  FOnInfoReceived := nil;
...
end; {Create}

procedure RecepcionDatos (Sender: TObject; Datos: pointer; NumDatos: integer);
begin
...
  // Procesamos los bytes que han llegado
...

  // Si todo ha ido bien generamos el evento que avisa de la
  // llegada de información útil }
  if Assigned (FOnInfoReceived) then
    FOnInfoReceived (Self, TramaRx.Info)
...

  // permitimos a las clases hijas procesar el evento OnDataReceived
  if Assigned(FOnDataReceived) then
    FOnDataReceived (Self, Datos, NumDatos);
end; {RecepcionDatos}
```

```
constructor TComEnlace.Create(AOw-
    ner: TComponent);
begin
...
  inherited OnDataReceived :=
    RecepcionDatos;
  FOnDataReceived := nil;
```

El evento que avisará y transportará la información útil, *OnInfoReceived*, se disparará sólo cuando hayan llegado datos y éstos sean correctos. De esta manera, el nivel superior, no tendrá que preocuparse nada más que de procesarlo (es

decir, cuando el nivel de enlace envía un *NACK* –ver Figura 2- o cuando se pierden tramas -Figura 3– y por tanto se producen reenvíos, no debe quedar constancia más que en el propio nivel de enlace).

OTROS EVENTOS NECESARIOS

Además del tipo de evento anterior que da el aviso de la llegada de información útil, necesitaremos definir varios eventos adicionales.

El primero de ellos avisará al transmisor de una trama, que ésta ha llegado correctamente al extremo receptor. Esto ocurrirá cuando llegue el *ACK* correspondiente a la trama enviada. Por tanto el evento que habrá que definir tendrá la siguiente forma:

```
TOnTxCorrect = procedure(Sender :
    TObject;) of object;
```

Será necesario otro evento para indicar que el medio de transmisión está roto o que el receptor está fuera de servicio. Si repetimos el reenvío de una misma trama un número de veces seguidas (que en principio hemos fijado con la constante *NUM_MAXIMO_REPETICIONES* en diez veces), damos por supuesto que la línea está rota o el receptor está apagado y avisamos al nivel superior con el siguiente evento:

```
TOnBrokenLine = procedure(Sender
    : TObject;) of object;
```

Un último evento que podrá ser útil definirá cualquier estado de error que se produzca en el nivel de enlace o cualquier error imprevisto en el protocolo y que en un principio dejaremos abierto. Para ello,

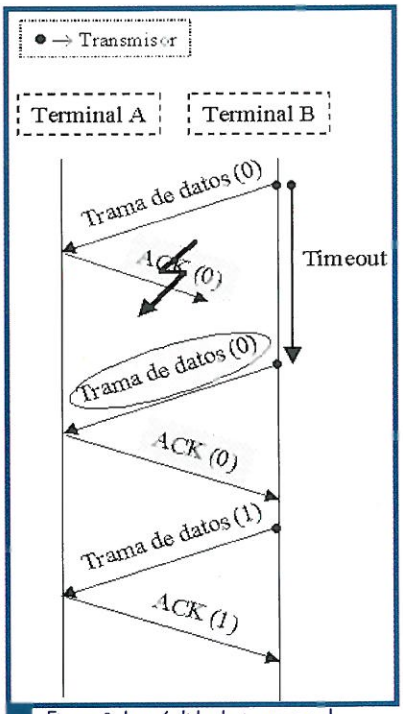


Figura 3.-La pérdida de tramas y el reenvío correspondiente, son procesos del nivel de enlace que deben quedar ocultos al nivel superior.

utilizamos un parámetro de tipo *integer* en el que codificaremos el tipo de error, de forma que quede abierto a futuras posibilidades de notificación de errores.

```
TOnComEnlaceError =
    procedure(Sender : TObject;
      TipoDeError: integer) of
    object;
```

La definición de todos estos tipos de eventos, así como su declaración dentro de la clase *TComEnlace* se muestra en el Listado 2.

DEFINICIÓN DE TIMEOUTS

Para terminar el apartado de definición de variables, necesitaremos una que controle el *timeout*, tanto el fijo como el variable. Esta variable será única y sólo variará la duración.

En un principio este intervalo tendrá un valor fijo (*TIMEOUT_TX*) y sólo cuando detectemos que los dos extremos de la comunicación están transmitiendo en el mismo instante de tiempo, emplearemos un valor aleatorio cuya duración mínima sea 0 milisegundos y su duración máxima sea:*TIMEOUT_TX_MAXIMO*.

Utilizaremos la clase *TTimer* para definir el *timeout*. La creación de este *timeout* y la inicialización de los valores que toma inicialmente se muestran a continuación:

```
TComEnlace = class(TComSerie)
private
  { Temporizador y manejador del
    temporizador }
  Timeout: TTimer;
  procedure TimeoutFinalizado
    (Sender: TObject);
...

constructor TComEnlace.Create(AOw-
    ner: TComponent);
begin
...
  Timeout:= TTimer.Create(Self);
  Timeout.Enabled:= False;
  Timeout.Interval:= TIMEOUT_TX;
  Timeout.OnTimer:= TimeoutFina-
    lizado;

estadoTx := etEsperoDatos;
end; {Create}

destructor TComEnlace.Destroy;
begin
  Timeout.Free;
  inherited Destroy;
end; {Destroy}
```

Como se puede observar, hemos añadido en el constructor la declaración de la variable **estadoTx** inicializándola con el valor **etEsperoDatos** ya que en un principio cualquiera de los dos extremos de la comunicación está en un estado de espera. Sólo cuando uno de los dos se decida a transmitir, pasará al

Listado 2: Definición y declaración del resto de eventos necesarios.

```
/ Aviso de que la trama ha llegado correctamente al otro extremo
  TOnTxCorrect = procedure(Sender : TObject;) of object;

// Evento que notifica que la línea está rota o que el receptor está apagado
TOnBrokenLine = procedure(Sender : TObject;) of object;

// Evento de aviso de cualquier error imprevisto en el protocolo
  TOnComEnlaceError = procedure(Sender : TObject; TipoDeError: integer) of object;

TComEnlace = class(TComSerie)
private
  FOnTxCorrect: TOnTxCorrect;
  FOnBrokenLine: TOnBrokenLine;
  FOnComEnlaceError: TOnComEnlaceError;

published
// Eventos para informar al nivel superior
  property OnTxCorrect: TOnTxCorrect
      read FOnTxCorrect write FOnTxCorrect;
  property OnBrokenLine: TOnBrokenLine
      read FOnBrokenLine write FOnBrokenLine;
  property OnComEnlaceError: TOnComEnlaceError
      read FOnComEnlaceError write FOnComEnlaceError;
...
end; { TComEnlace }

...

constructor TComEnlace.Create(AOwner:TComponent);
begin
...
  FOnTxCorrect := nil;
  FOnBrokenLine := nil;
  FOnComEnlaceError := nil;
...
end;
```

Listado 3: Procedimiento de recepción de datos y procesado de las tramas.

```
procedure RecepcionDatos (Sender: TObject;
    Datos: pointer; NumDatos: integer);
const cuenta: integer = 0;

{ ------- procedimiento anidado -------- }
procedure ComEnlace.ProcesarTrama;
var
  pRecibido: PChar;
  Recibido: string;
  i, j: integer;
  const num: integer = 0;

begin
  pRecibido := Datos;
  Recibido := '';
  i:= NumDatos
  while i > 0 do
    begin
      Recibido := Recibido + pRecibido^;
      inc(pRecibido);
      dec(i);
    end;
  i:= 0;
  if (cuenta = 1) then begin
      inc(cuenta);
      inc(i);
      TramaRx.Tipo := Ord(Recibido[i]);
    end;

  if (cuenta = 2) then begin
    inc(cuenta);
    inc(i);
    TramaRx.NumSec := Ord(Recibido[i]);
end;

if (cuenta = 3) then begin
  inc(cuenta);
  inc(i);
  TramaRx.Length := Word(Ord(Recibido[i]));
end;

if (cuenta = 4) then begin
  inc(cuenta);
  inc(i);
  TramaRx.Length := TramaRx.Length +
      256*(Word(Ord(Recibido[i])));
  dec(NumDatos);
end;

  while (num < integer(TramaRx.Length) - 1)
and
        (NumDatos – i > 0) do begin
  inc(cuenta);
  inc(i);
  TramaRx.Info[num] := Recibido[i];
  inc(num);
end;

if (cuenta = 5 + TramaRx.Length) then
  TramaRx.CRC :=
      Word(Ord(Recibido[5 +
TramaRx.Length]));

if (cuenta = 6 + TramaRx.Length) then
  begin
  TramaRx.CRC := Word(Ord(Recibido[6 +
      TramaRx.Length]));
  estadoRecepcion:=erEsperaPrincipioTrama;

  ActuarSobreTrama;
  end;

end;
{ ------- fin procedimiento anidado -------- }

begin
  case estadoRecepcion of
    erEsperaPrincipioTrama:
      case byte(Ord(PChar(Datos)[0])) of
        PRINCIPIO_TRAMA:
          begin
            erEsperaDatos;
            cuenta := 0;
            ProcesarTrama;
          end;
      else
        { error };
      end; {case}
    erEsperaDatos:
      ProcesarTrama;
    end;

// permitimos a las clases hijas procesar el evento "OnDataReceived"
if Assigned(FOnDataReceived) then
    FOnDataReceived (Self, Datos, NumDatos);
end; {RecepcionDatos}
```

estado de "espera de *ACK*", con lo que esta variable pasará a tomar el valor **etEsperoAck.**

RECEPCIÓN DE DATOS

El procedimiento *RecepcionDatos*, que se activará tras la llegada de cualquier *byte* al puerto serie, se encargará de procesar la información que vaya llegando al puerto. Este procesamiento incluye:

- **Delimitación de las tramas**: la información llega al puerto en forma de *bytes*. Debemos distinguir y agrupar los *bytes* que llegan en las tramas a que corresponden.
- **Procesamiento de las tramas**: una vez delimitadas deberá realizar la asignación de los *bytes* de la trama a los campos de control y de información correspondientes.
- **Generación de eventos**: tanto si la trama es de datos como si es de *ACK* deberá generar el evento correspondiente. Es decir, cuando llegue una trama de información útil y ésta sea correcta (el *CRC* sea correcto) generará el evento *OnInfoReceived*. Cuando llegue una trama de *ACK* generará el evento *OnTxCorrect*.

Para realizar la primera función, deberá comprobar primero, para todos los *bytes* que van llegando, si alguno corresponde al *byte*

identificado como *flag* (*PRINCIPIO_TRAMA*).

Una vez identificado este primer *byte* deberá conocer cuántos *bytes* han llegado de la trama correspondiente, ya que puede ocurrir que lleguen en varios trozos.

Para controlar este estado de recepción de los datos definimos una nueva variable que tome cuenta de este "estado de la recepción". De esta forma, al principio estaremos en el estado de espera del principio de la trama y en cuanto llegue el *flag*, pasaremos a esperar el resto de datos.

```
TEstadoRecepcion =
  (erEsperaPrincipioTrama, erEs-
    peraDatos);
```

Para conocer el número de *bytes* que han llegado de una trama (y por tanto, los que quedan por llegar), utilizaremos una variable estática en el procedimiento *RecepcionDatos* que lleve la cuenta del número de *bytes* de la trama que han entrado por el puerto serie. A esta variable la denominaremos **cuenta**.

Como la recepción de las tramas se puede producir de forma troceada deberemos disponer de un contador adicional, que almacenenará - en caso de recepción en varias partes - la posición en la que nos hemos quedado dentro de la trama. Esta otra variable la denominaremos **num**.

Dentro del procedimiento *RecepcionDatos*, crearemos un procedimiento anidado (para que pueda aprovechar la variable estática **cuenta**), en el que se realice el procesamiento de los *bytes* y la segmentación de las tramas. Una vez, que tengamos una trama completa llamaremos a un procedimiento cuya función consistirá en decidir qué se debe hacer en función del protocolo definido.

En el procedimiento *RecepcionDatos*, si estamos en el estado inicial de "espera de principio de trama" se recoge el primer *byte* recibido y se compara con el *byte* de inicio de trama (**01111110**). Si coincide, comenzamos a procesar el resto de los *bytes* mediante el procedimiento *ProcesarTrama*. Por tanto el código necesario será :

Figura 4.- Con el protocolo de nivel de enlace conseguiremos transmisiones fiables punto a punto, independientemente de la distancia entre ellos.

```
case estadoRecepcion of
  erEsperaPrincipioTrama:
   case byte(Ord(Datos[0]) of
     PRINCIPIO_TRAMA:
     begin
       estadoRecepcion := erEspe-
    raDatos;
       cuenta := 0;
       ProcesarTrama;
     end;
     else
       { error };
   end; {case}
  erEsperaDatos:
      ProcesarTrama;
end;
```

Además cambiamos el valor de la variable *estadoRecepcion* para que pase a esperar datos en lugar del principio de la trama.

En cuanto al procedimiento *ProcesarTrama*, realizará el rastreo de los *bytes* que lleguen y los irá asignando a las variables de la estructura *TramaRx*. En este procedimiento se debe tener en cuenta a la hora de asignar valores a las distintas variables pues es posible que se ejecute varias veces para una sola trama. Por tanto, debe asegurar que no asigna valores erróneos a las variables de control y a la variable que guarda la información útil.

Una vez que llegue el último *byte* de la trama (el segundo *byte* que forma el *CRC*), cambiaremos el estado de recepción de nuevo a *erEsperaPrincipioTrama*, y llamaremos a la función *ActuarSobreTrama* que se encargará de decidir, cuál será el siguiente paso que se debe tomar.

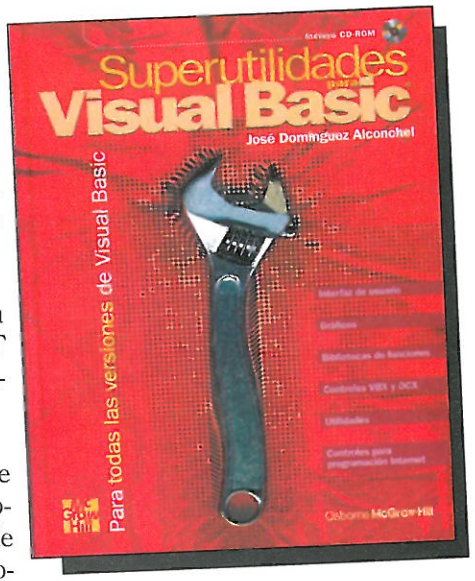
El procedimiento *RecepcionDatos* se muestra completo en el Listado 3, donde se puede observar el procedimiento *ProcesarTrama* anidado del mismo.

CONCLUSIÓN

En este artículo hemos visto las constantes, tipos, eventos y variables necesarias para el desarrollo del componente *Delphi*. Además hemos desarrollado la función de recepción y tratamiento de los datos. Aún quedan pendientes, la función que se encarga de automatizar la actuación del componente, en la definición de los procedimientos de envío de las tramas.

En el próximo artículo terminaremos la implementación del componente *TComEnlace* y realizaremos una aplicación que sirva para transmitir ficheros y mensajes, que además de útil, servirá para comprobar el correcto funcionamiento del componente.

SUPERUTILIDADES PARA VISUAL BASIC



Este es un manual que recoge técnicas, trucos y utilidades para programar eficientemente con *Visual Basic*. A lo largo de sus páginas el lector encontrará multitud de ejemplos de programación detallados y explicados paso a paso, así como soluciones profesionales a los problemas más frecuentes que aparecen cuando se programa en este lenguaje.

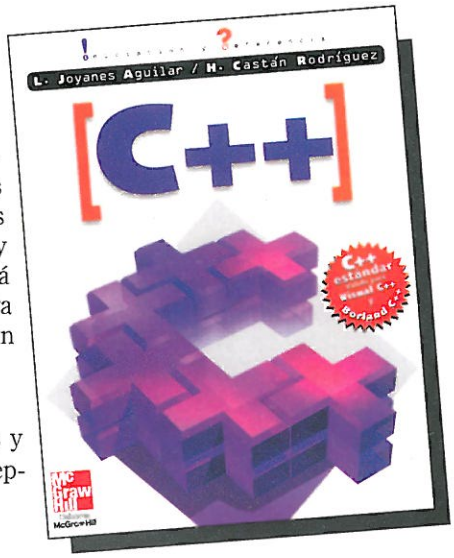
El libro abarca una amplia gama de epígrafes que incluyen temas como: la programación en 32 *bits*, diferencias entre *Windows* 95, *NT* 3.51 y *Windows NT* 4.0, la interfaz de usuario, gráficos, multimedia, bibliotecas de funciones, controles *VBX* y utilidades complementarias.

Este volumen aparece acompañado de un *CD-ROM* complementario que se encarga de aportar todos los ejemplos y código fuente del libro, descompiladores, protectores de programa, cientos de imágenes para utilizar en botones de barras de herramientas, controles para programaciones en *Internet* o analizadores de proyectos *Visual Basic* entre otras utilidades.

Si lo que pretende es aprender a programar en Visual Basic, este manual le ayudará a conseguir su meta gracias a unas explicaciones sencillas que eliminarán cualquier duda que se le pueda plantear.

Editorial: Mc Graw Hill
Nº de Páginas: 592
Nivel: Intermedio
Autor: José Dominguez Alconchel
Idioma: Español
Precio: 7.900 (I.V.A. inc.)

C++



C++ está considerado como el lenguaje de programación por excelencia para el desarrollo de aplicaciones orientadas a objetos y además probablemente el más popular. En este libro encontrará todos los pasos para iniciarse y adquirir soltura en la programación en *C++* estándar y por tanto en cualquier compilador. A través de 10 capítulos conocerá todos los conceptos de este lenguaje de programación de una manera progresiva, desde los elementos básicos hasta los conceptos que dan lugar a temas avanzados.

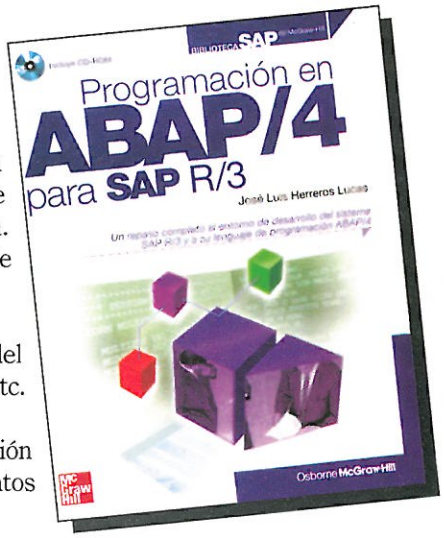
Aprenderá sobre datos, sentencias, funciones, estructuras, uniones y punteros, para luego pasar al tratamiento avanzado de funciones, conceptos de programación orientada a objetos, clases y herencia.

La parte final del manual cuenta con una completa referencia de los menús de los *IDE* de *Visual C++* 6.0 y *Borland C++ Builder* 3 y 4, además de una extensa lista de recursos para conocer a fondo todos los secretos de *C++*.

Se trata de un método de aprendizaje rápido y cómodo que cuenta con un buen número de ejemplos y figuras para una asimilación visual de conceptos, a los que también contribuyen los iconos de consulta rápida.

Editorial: Mc Graw Hill
Nº de Páginas: 367
Nivel: Básico-Intermedio
Autor: L. Joyanes Aguilar, H. Castán Rodríguez
Idioma: Español
Precio: 3.995 (I.V.A. inc.)

PROGRAMACIÓN EN ABAP/4 PARA SAP R/3



ABAP/4 (*Advanced Bussines Application Programming* / 4) es el lenguaje de programación con el que está desarrollado el sistema integrado de gestión *SAP/R3,* existe un entorno de desarrollo *ABAP/4* completo, que permite ampliar o modificar la funcionalidad del sistema. Este libro describe tanto el entorno de desarrollo del sistema *SAP/R3* como el lenguaje de programación *ABAP/4*.

A lo largo de los capítulos el desarrollador conocerá la descripción de las transacciones del entorno de programación: Editor *ABAP/4, Screen, Painter*, Menú *Painter*, Diccionario de datos, etc.

Se hace un repaso de la definición de los conceptos básicos del lenguaje de programación *ABAP/4:* reglas sintácticas del lenguaje, declaración de tipos y de objetos de datos, procesamientos de tablas internas, pantalla de selección, acceso a la base de datos, programas interactivos, etc.

Los conceptos más avanzados también tienen cabida entre las páginas de este libro, así el lector descubrirá contenidos como la depuración de un programa, la *interfaz* de datos y comunicaciones, generación dinámica de objetos, *logs* de aplicación, procesos de fondo o gestión de errores entre otros muchos.

Editorial: Mc Graw Hill	Autor: José Luis Herreros Lucas
Nº de Páginas: 700	Idioma: Español
Nivel: Experto	Precio: 6.900 (I.V.A. inc.)

SÓLO PROGRAMADORES

DESARROLLO DE APLICACIONES LINUX CON GTK+ Y GDK

Con el explosivo crecimiento de *Linux* en los últimos años, la demanda de aplicaciones de calidad nunca ha sido mayor, esta guía avanzada satisface ampliamente esta creciente necesidad, ilustrando la multitud de herramientas y tácticas de *GTK+* y *GDK* que un desarrollador *Linux* necesita para crear aplicaciones con interfaz gráfica de usuario.

Orientado hacia los programadores que posean amplios conocimientos de *C*, este libro enseña el tema a través de ejemplos detallados.

El autor incorpora los fundamentos de *GTK+* en los ejemplos para proporcionar al lector una experiencia de aprendizaje detallada. Después, se encarga de llevar al lector al siguiente nivel de desarrollo, proporcionando información sobre cómo crear *widgets* originales, cómo diseñar entornos gráficos complejos con *GDK* y cómo trabajar con *sprites* y animaciones.

Gracias a esta guía será capaz de dar color y vida a sus aplicaciones con *GDK*, comprender *GLIB*, aprender trucos y atajos para mejorar sus capacidades de programación en *C*, entre otros contenidos.

Perfecto para el desarrollador experimentado en entornos *Windows* o *UNIX* que esté interesado en aprender sobre *GTK+*, este manual será el encargado de reforzar esa experiencia.

Editorial: Prentice Hall	Autor: Eric Harlow
Nº de Páginas: 482	Idioma: Español
Nivel: Experto	Precio: 4.500 (I.V.A. inc.)

FE DE ERRATAS: En el nº 61 de SÓLO PROGRAMADORES se produjo un error al incluir el precio del libro SQL SERVER 7. Su precio real es de 8.500 ptas. (I.V.A. INC.)

Dudas técnicas

Cualquier duda o cuestión que queráis plantear tiene cabida en esta sección que confeccionáis vosotros mismos, los lectores de SÓLO PROGRAMADORES.

PREGUNTA

Amigos de *Sólo Programadores*:

Lo primero que quiero hacer es felicitaros por la estupenda revista que publicáis. El incluir fotografías y dibujos en color le da una pinta mucho más profesional y nos la hace más agradable a todos.

Ahí va mi pregunta, si es demasiado sencilla, o no creéis que merezca la pena publicarla, intentad contestármela a través de correo electrónico. Yo soy usuaria de *Windows 95*. A través de la Red he podido encontrar numerosos temas de *Windows*, en los que se modifican las pantallas de inicio y cierre de *Windows 95*.

Supongo que hará falta algún programa para modificarlas o para cambiar la imagen. Yo tengo *Photoshop* y he preparado una serie de imágenes que me gustaría que aparecieran al arrancar y apagar el sistema. ¿Me podéis indicar los pasos necesarios para cambiar las fotos?

Muchas gracias por adelantado.

Carmen P. D. Madrid

RESPUESTA

Estimada Carmen:

Lo primero agradecerte tus comentarios y tus palabras de ánimo. Aunque nos debemos a vosotros, nuestros lectores, muchas veces no nos damos cuenta de que hay aspectos que aún podemos mejorar, y es lo que nosotros queremos seguir haciendo.

En cuanto a tu pregunta relativa a las imágenes que *Windows 95* ofrece al arrancar y apagar el sistema, vamos a intentar responderte con la mayor precisión posible.

Las tres imágenes especiales que podemos contemplar al arrancar y apagar *Windows 95* responden a los siguientes nombres:

```
logo.sys
logos.sys
logow.sys
```

La primera de ellas es un fichero oculto en el directorio raíz de nuestro disco duro *(C:\>)*, mientras que las otras dos las podemos encontrar en el directorio de *Windows 95* que es, típicamente, *C:\WINDOWS>*.

Figura 1.- Los ficheros *logos.sys* y *logow.sys* son las imágenes que nos muestra nuestro sistema al apagar la máquina.

Para cambiar las imágenes de arranque y final de sesión, los diseñadores de los diferentes temas de *Windows 95* se limitan a sustituir estos ficheros por otros más acordes con sus creaciones.

En tu caso, bastará con que sustituyas estas imágenes por las tuyas, teniendo en cuenta que deben estar en un formato determinado. Las características que han de tener las imágenes son las siguientes:

- **Formato:** *MS Windows BMP*
- **Profundidad:** 8 *bits por pixel* (256 colores)
- **Ancho:** 320
- **Alto:** 400

Para cumplir estas condiciones, puedes utilizar el *Photosop* para comprimir o expandir la imagen a tu gusto, recordando siempre que tendrás que salvarla con el nombre antes mencionado, aunque el formato sea *BMP*.

PREGUNTA

Estimados amigos:

Soy un estudiante de segundo de Informática de Madrid y me dirijo a vosotros para ver si me podéis contestar esta pregunta.

Hace algún tiempo que estoy utilizando una serie de programas que sólo se pueden usar en

MS-DOS porque parece que están en 'modo protegido' o algo así. El caso es que ahora voy a tener que utilizarlos varias veces seguidas apagando el *PC* entre medias y me resulta muy incómodo arrancar *Windows 98*, esperar a que empiecen y rearrancar la máquina en modo *MS-DOS*.

He intentado ejecutarlos en ventana, pero me da un "Error de protección general" en el módulo "*Kernel-32.dll*". ¿Se puede hacer algo para que el sistema empiece en *MS-DOS* como lo hacía en *Windows 3.1*?

Gracias por vuestra respuesta.

Pedro E. M. Madrid.

RESPUESTA

Estimado Pedro:

El error de protección general ocurre en tu máquina cuando ejecutas un programa de *MS-DOS* que se coloca en modo protegido. En todo procesador 386 o superior existe un modo especial del mismo, denominado "protegido" o "supervisor", en el que se tiene control absoluto sobre todos los parámetros del sistema.

Una de las peculiaridades de este "modo protegido" es la de que, en un momento determinado, sólo un programa tiene acceso a él. En el caso de Windows es el núcleo del sistema el que se ejecuta en "modo supervisor", por lo que ninguna otra aplicación podrá utilizarlo.

Los demás programas de *Windows* deberán utilizar las *API's (Application Program Interface)* de 16 ó 32 *bits* denominadas *Win16* y *Win32*, mientras que las aplicaciones *MS-DOS* se ejecutarán en máquinas virtuales *x86* en lo que se llama modo *Virtual86 (V86)*.

En cuanto a tu necesidad de botar directamente en *MS-DOS*, con todos los *drivers* cargados y sin necesidad de rearrancar el sistema, ni darle a *F5/F8* en el botado, nuestra solución es:

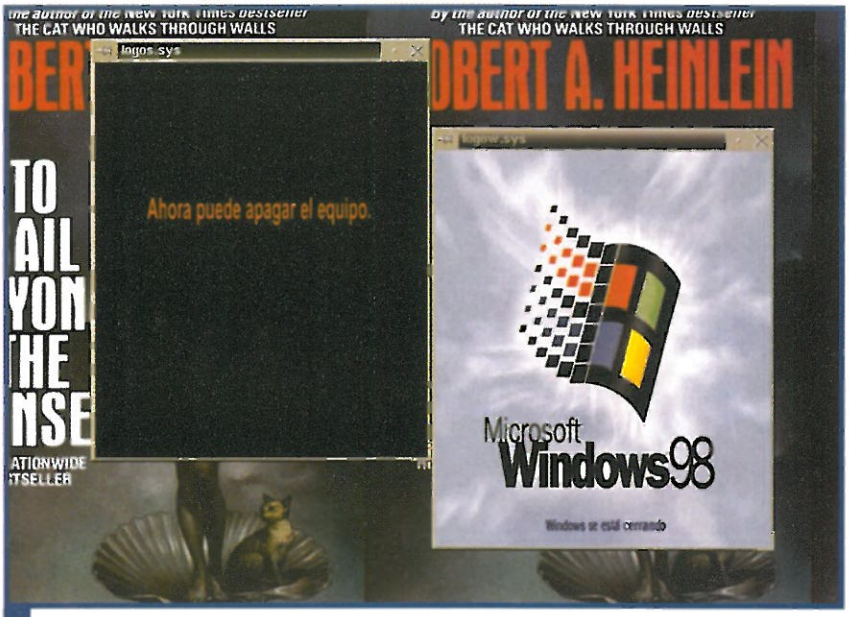


Figura 2.- El contenido del fichero MSDOS.SYS determina varios parámetros de arranque del sistema.

Lo primero buscar el fichero oculto llamado *MSDOS.SYS* que podemos encontrar en el directorio raíz del disco *C:*. Para ello, abriremos una ventana de *MS-DOS* y después de colocarnos en el citado directorio, desprotegeremos el fichero con la instrucción:

```
C:\> attrib -s -h -r  msdos.sys
```

Realizada esta operación podremos editar el archivo, bien con el *NOTEPAD* o bien desde *MS-DOS* directamente con el comando *EDIT*.

Una vez dentro del fichero, podremos observar la siguiente secuencia:

```
;SYS
[Paths]
WinDir=C:\WINDOWS
WinBootDir=C:\WINDOWS
HostWinBootDrv=C

[Options]
BootMulti=1
BootGUI=1
DoubleBuffer=1
AutoScan=1
WinVer=4.10.1998
;The following lines are required
    for compatibility with other
    programs.
;Do not remove them (MSDOS.SYS
    needs to be >1024 bytes).
```

Estas líneas definen varios parámetros generales de nuestra máquina. En concreto, la instrucción que nos interesa es la segunda del apartado denominado *OPTIONS*. En esta línea podemos encontrar el comando:

```
BootGUI=1
```

El significado de esta directiva es que, en el arranque de la máquina *(Boot)*, se ejecute automáticamente la interfaz gráfica de usuario *(GUI)*. Por lo tanto, bastará con que cambiemos su valor de 1 (cierto) a 0 (falso).

```
BootGUI=0
```

Una vez realizado el cambio guardaremos el archivo para, a continuación, volver a protegerlo con el comando:

```
C:\> attrib +s +h +r msdos.sys
```

La siguiente vez que arranquemos nuestra máquina, tendremos al viejo conocido *PROMPT* de *MS-DOS* a nuestra total disposición.

Para concluir queremos indicar que la orden *WIN* en línea de comandos, arranca el sistema como en versiones anteriores.

PREGUNTA

Un saludo para todos los lectores:

He estado leyendo vuestros últimos números y he podido ver, con gran sorpresa por mi parte, que habéis vuelto a poner una parte de *Linux*. Hacía ya tiempo que no veía nada y pensaba que quizás os habíais olvidado de este sistema.

Yo no he trabajado nunca en el *Linux* ese, pero estoy muy interesado en empezar a hacerlo. El caso es que tengo un único disco duro con una única partición en la que tengo aproximadamente un *Giga* libre de espacio.

Como no me gustaría tener que pasar otra vez por una reinstalación de *Windows*, preferiría no tener que volver a reformatear mi disco. Así que mi pregunta va en ese sentido:

¿Cómo podría utilizar *Linux* como un subprograma de *Windows*, o a través de *Windows*, ya sea en una ventana, o desde *MS-DOS*, de tal manera que no tenga que reformatear mi disco duro y no me estropee la información que éste contiene?

En fin, quedo a la espera de vuestra respuesta.

RESPUESTA

Estimado lector:

Lo primero que desde *Sólo Programadores* queremos hacer es informarte que la ausencia de *Linux* en nuestra revista en este tiempo atrás ha sido debida al éxito de *Sólo Programadores Linux*, la cual ya lleva 13 números en el mercado y está disfrutando del interés de sus lectores.

En cualquier caso, y dado que el tema de *Linux* es de profunda actualidad, hemos decidido continuar tratándolo en *Sólo Programadores*.

En cuanto a tu pregunta, la respuesta es NO. Es imposible ejecutar *Linux* y *Windows 95/98/NT* en la misma máquina, ya que ambos sistemas operativos requieren el control completo del *hardware* del sistema. Como comentábamos en una de las preguntas anteriores, sólo un programa puede estar en modo protegido en un momento dado y, tanto el *kernel* de *Linux* como el núcleo de *Windows* necesitan ser la aplicación afortunada.

Sin embargo, los diseñadores de *Linux* han tenido en cuenta tu problema. Hay mucha gente que desea probar *Linux* o tenerlo ocasionalmente instalado en su *PC*, pero sin por ello tener que renunciar definitivamente a una serie de particiones.

El truco consiste en instalar *Linux* en la misma partición que tienes las *Windows* en un directorio que se podría llamar, por ejemplo *C:\LINUX>*.

Para ello, en cualquier distribución común de *Linux, (RedHat, SuSe, SlackWare, Caldera,Debian,* etc.) tienes la posibilidad de utilizar un sistema de ficheros denominado *UMSDOS*.

Este sistema de ficheros tiene las siguientes ventajas:

- Permite a *Linux* y *Windows* coexistir pacíficamente en una única partición.
- No necesita redistribuir el espacio de disco ni rehacer particiones.

Como desventaja fundamental, *UMSDOS* al ir sobre *FAT*, el sistema de ficheros de *MSDIS/Windows*, es un orden de magnitud más lento que *EXT2*. Sin embargo hay veces que la versatilidad que nos ofrece merece la pena frente a la pérdida de velocidad.

Finalmente, te aconsejamos desde aquí que busques en la Red alguna distribución que te permita instalarlo de este modo, directamente desde *Windows*. De esa manera lo tendrás enormemente fácil. Con el *CD* que acompaña a la revista puedes probar una de estas distribuciones.

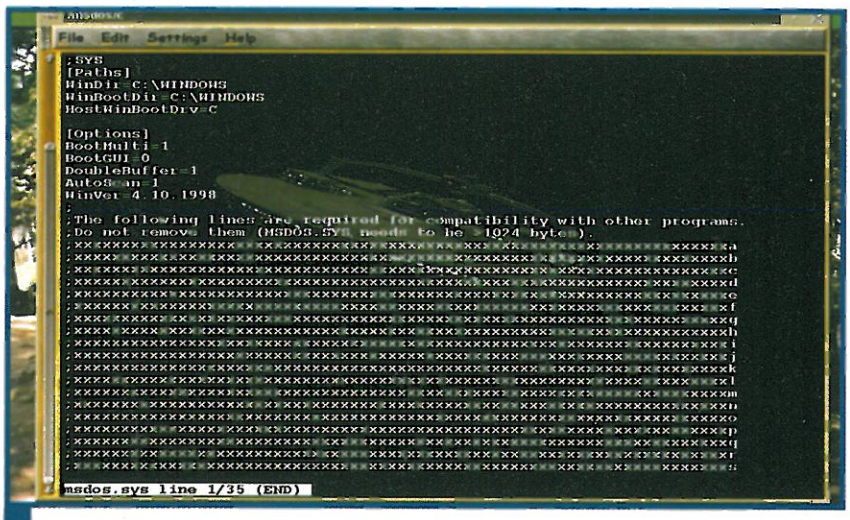


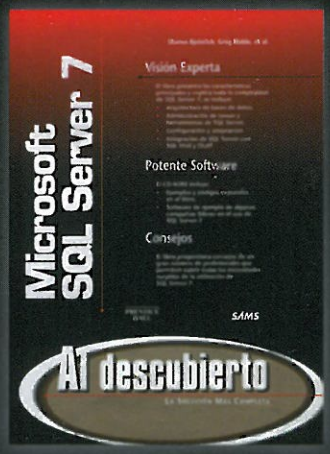
Figura 3.- Es posible tener Linux y Windows en la misma partición de nuestro PC gracias al sistema de ficheros UMSDOS.

TODOS LOS MESES EN TU QUIOSCO

PARA ESTAR AL DÍA EN SOFTWARE SIN GASTARSE UN EURO EN TELÉFONO

- LOS MEJORES ESPECIALES
- MULTIMEDIA
- NEGOCIOS
- DISEÑO
- UTILIDADES
- LO ÚLTIMO
- INTERNET
- EN CASTELLANO
- Y MUCHO MÁS...

Libros para profesionales escritos por profesionales

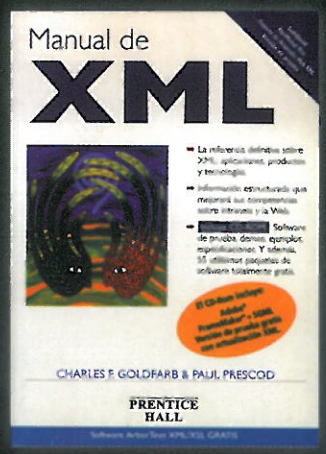


Incluye CD-Rom

Microsoft SQL Server 7
Al Descubierto
Sharon Bjeletich et al.

Edición Especial
TCP/IP
John Ray

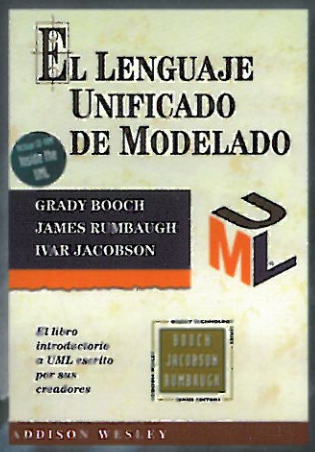
Incluye CD-Rom



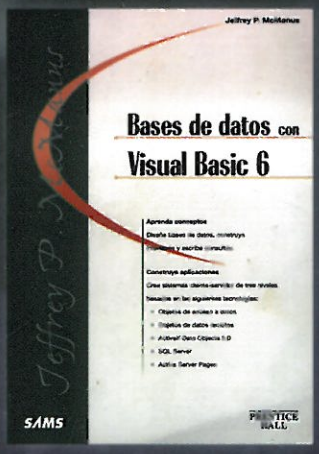
Incluye CD-Rom

Manual de XML
Charles F. Goldfarb
& Paul Prescod

Incluye CD-Rom



UML. El Lenguaje Unificado de Modelado
Grady Booch et al.



Bases de Datos con Visual Basic 6
Jeffrey P. McManus

Incluye CD-Rom

¡Ya a la venta en tu librería habitual!

Prentice Hall

Líder mundial en libros y recursos de informática